Vol. 13, No. 11 • November 1997

For Macintosh Programmers & Developers

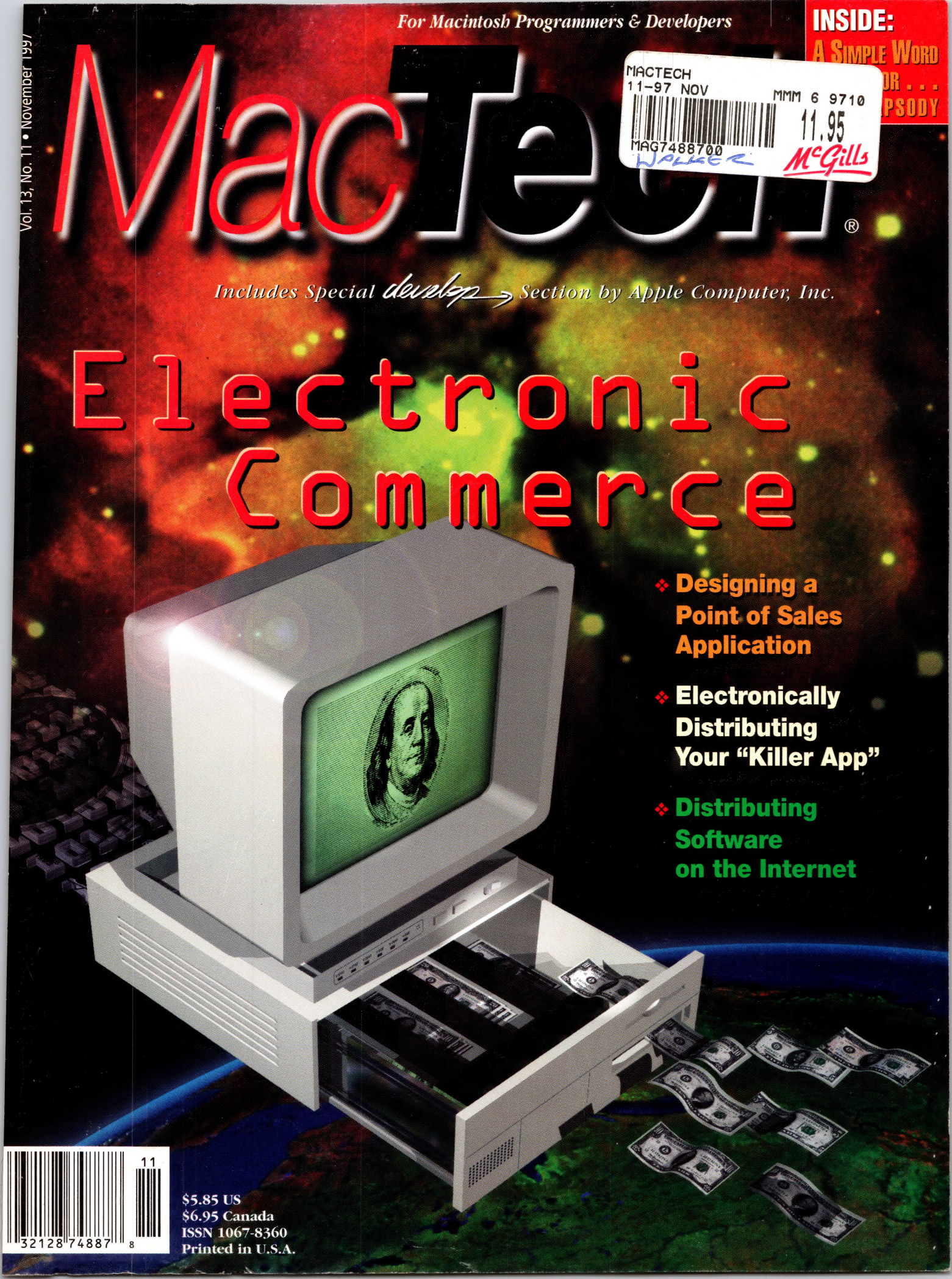
MacTech®

INSIDE:
A SIMPLE WORD
OR . . .
APSODY

MACTECH
11-97 NOV
MMM 6 9710
MAG7488700
11.95
WALKER
McGills

Includes Special develop Section by Apple Computer, Inc.

Electronic Commerce

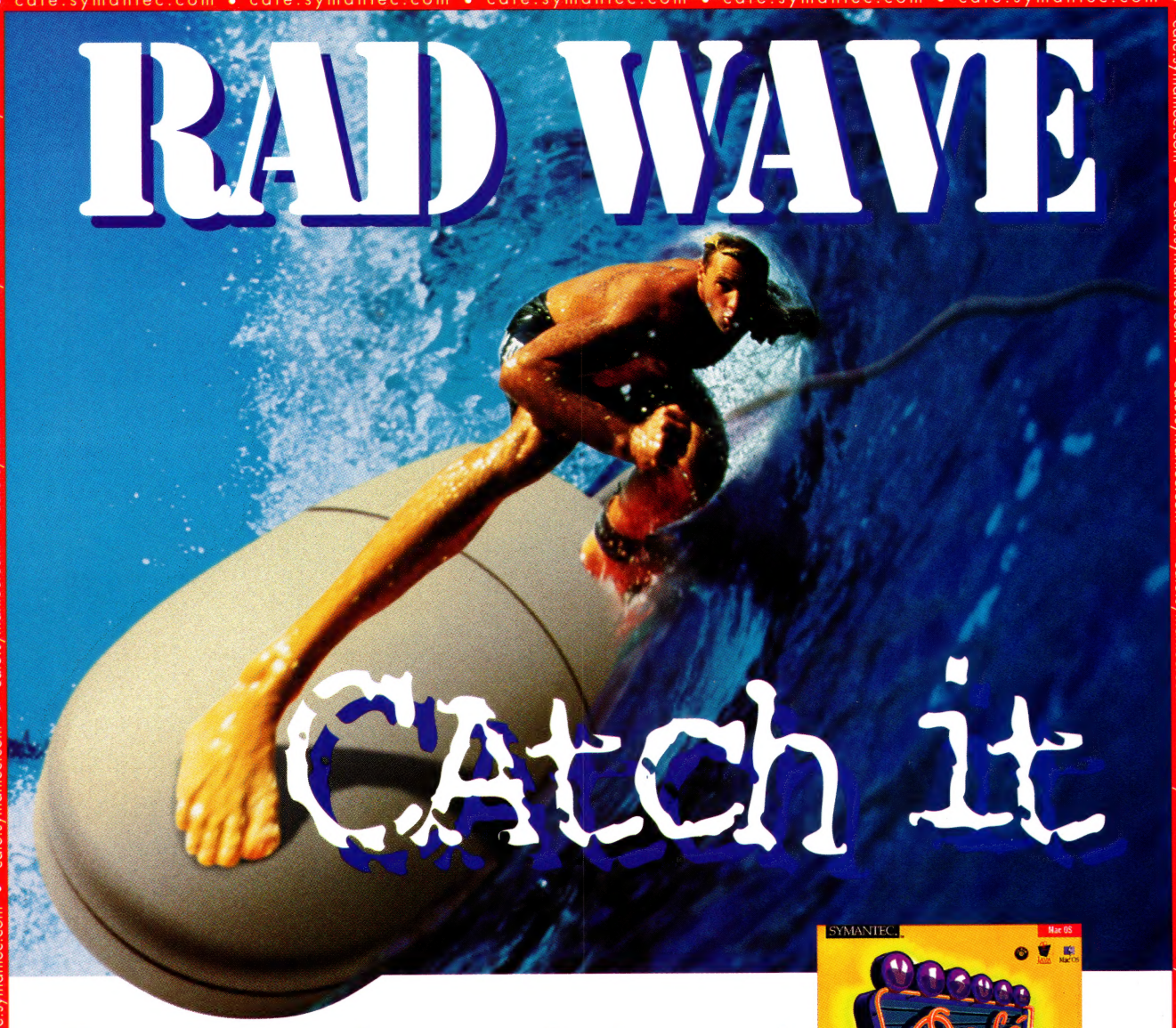


- Designing a Point of Sales Application
- Electronically Distributing Your "Killer App"
- Distributing Software on the Internet

11
32128 74887 8
$5.85 US
$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

• cafe.symantec.com • cafe.symantec.com • cafe.symantec.com • cafe.symantec.com • cafe.symantec.com •

RAD WAVE

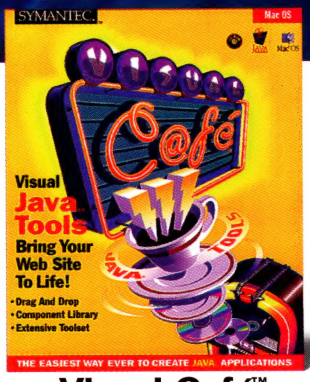


Catch it

Develop web apps faster and easier with the only RAD Java™ tool for the Mac!

Symantec's Visual Café.™ The *only* Rapid Application Development (RAD) Java tool for the Mac. Guaranteed to let you knock out dynamic web apps at tsunami speed, with drag-and-drop ease. And best of all, your apps are ready to run on Rhapsody. So why use anything else? Call or visit our website today!

Mac OS



Visual Café™

Write in Java today, it runs in Rhapsody tomorrow!

JDJ Java Developer's Journal World Class Award

winner The cnet Award for Internet Excellence 1997 best Internet development tool

Software Development Product Excellence Award 1996

Call for a FREE trial copy!

1-800-453-1059ext 9H32

Or download from:

http://cafe.symantec.com

SYMANTEC.

Symantec and the Symantec logo are registered trademarks and Symantec Visual Café is a trademark of Symantec Corporation. Java is a trademark of Sun Microsystems, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. Apple, Macintosh and Power Macintosh are registered trademarks of Apple Computer, Inc. All other brand names or trademarks are the property of their respective owners. ©1997 Symantec Corporation. All rights reserved. In Canada, call 1-800-365-8641. In Australia, call 02-9850-1000. In Europe, call 31-71-535-3111. Free trial copies of Visual Café, while supplies last.

• cafe.symantec.com • cafe.symantec.com • cafe.symantec.com • cafe.symantec.com • cafe.symantec.com •

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

For Macintosh Programmers & Developers

MacTech MAGAZINE

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@devdepot.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

MACTECH MAGAZINE

MacTech Magazine is grateful to the following individuals who contribute on a regular basis. We encourage others to share the technology.

We are dedicated to the distribution of useful programming information without regard to Apple's developer status.

For information on submitting articles, ask us for our ***writer's kit*** *which includes the terms and conditions upon which we publish articles.*

Editorial Board of Advisors

Scott T Boyd, Jordan J. Mattson, Jim Straus, and Jon Wiederspan

Editors

Publisher • Neil Ticktin
Editor-in-Chief • Eric Gundrum
Managing Editor • Jessica Courtney
Online Support • Nick "nick.c" DeMello

Contributing and Technical Editors

Components • Tantek Çelik, Microsoft Corporation
Internet • Carl de Cordova, Apple Computer, Inc.
Java • Will Iverson, Apple Computer, Inc.
MacDev-1™ • Rich Siegel, Bare Bones Software, Inc.
Mac OS 8 • Steve Kiene, Mindvision
MagicCap/TeleScript • Richard Clark
Performance Programming • Jim Gochee, Connectix
Product Reviews • Ed Ringel
Rhapsody • Michael Rutman
Miscellaneous Topics • Jeff Ganyard

Regular Columnists

Getting Started • Dave Mark
Programmer's Challenge • Bob Boonstra
From the Factory Floor • Dave Mark, Metrowerks
Tips & Tidbits • Steve Sisak
MacTech Online • Nick "nick.c" DeMello

XPLAIN CORPORATION

Chief Executive Officer • Neil Ticktin
Chief Operating Officer • Andrea J. Sniderman
Controller • Heather Principe
Advertising Executive • Ruth Subrin
MarComm Manager • Susan M. Whitney
Events Manager • Susan M. Worley
Network Administrator • Chris Barrus
Customer Relations • Erik Davidson, Stephanie Long, Lee Ann Pham, Susan Pomrantz
Accounting • Jan Webber

Art Direction/Production • InfoGraphix
Board of Advisors • Steven Geller, Blake Park, and Alan Carsrud

PRINTED WITH SOY INK™

This publication is printed on paper with recycled content.

All contents are Copyright 1984-1997 by Xplain Corporation. All rights reserved. MacTech is a registered trademark and MacDev-1, THINK Reference, Developer Depot, Sprocket, JavaTech, WebTech, BeTech, and the MacTutorMan are trademarks of Xplain Corporation. *develop* is a trademark of Apple Computer, Inc. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders. Xplain Corporation does not assume any liability for errors or omissions by any of the advertisers, in advertising content, editorial, or other content found herein. Opinions or expressions stated herein are not necessarily those of the publisher and therefore, publisher assumes no liability in regards to said statements.

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

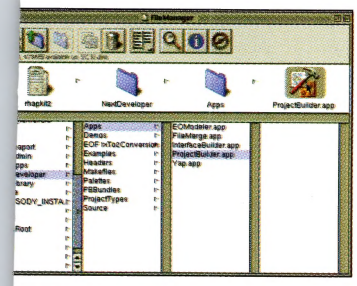
POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

For Macintosh Programmers & Developers

MacTech® MAGAZINE

Contents

November 1997 • Volume 13, No. 11



A Simple Word Processor... On Rhapsody, page 12

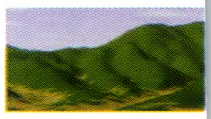
Feature Articles

Reader Resources

Columns

Designing a Point of Sales Application, page 22

develop Special Section



by Eric Gundrum

Why All This Crypto Stuff?

You can't have digital commerce unless you can be certain that your digital data is unadulterated. That's where cryptography comes in. Most people think of cryptography simply as a means to hide data, but actually it is much more useful than that. Through cryptography, we can assure (within reasonable expectations) that a block of data has not been changed by any intermediate party.

Imagine that you are a software publisher, and you want to use the Internet to distribute an update to your software. You could post the update to a few key software distribution sites and let it propagate. Within a day or two everyone would have access to it, but how can you be certain that it was your version of the update that was distributed? What's to stop some malicious person from releasing his own version of your updater and embedding his newly created virus? (Many readers may recall a similar situation with the recent release of a bogus StuffIt Deluxe 4.5 package as a TROJAN HORSE.) This problem can easily be prevented through the proper use of cryptography; in this case, digital signatures.

Another use of cryptography is access certificates. Imagine selling your software over the 'Net: your potential customer begins by examining a trial version of your software. The customer decides to purchase the software and sends you payment through some digital means. (Payment could be digital cash, digital checks, credit card information through an SSL connection, or some other means.) Once you have settled the payment with a bank (to be sure it is not fraudulent), you send an authentication code to the customer so they can turn the trial version of the software into the fully paid version. What is that authentication code? Many software companies struggle for countless hours trying to develop a serial number that can't be easily guessed or changed, and contains enough information to be traceable back to the customer in case they give it out for their friends to use. Another limitation of this mechanism is that you, the publisher, must maintain a database to map all the serial numbers to customers. This becomes even more complicated when you have single customers purchasing more than one copy of the product, or add site licenses, multiple versions, or other products.

Access certificates eliminate most of the limitations of serial numbers. An access certificate is a document which contains all the information the software publisher uses to control access to the software. This can include the purchaser's name and contact information and various license restrictions such as how many copies can be used simultaneously or when this use of the software expires. Cryptography is used to digitally sign the access certificate; the software being accessed can check this signature to verify the document has not been altered. The software can read the certificate to determine what restrictions to impose on the use of the software. If the information in the certificate is stored as clear text, then the customer also can see what restrictions are on the license. This certificate serves much the same purpose as a serial number; however, users are much less likely to distribute a certificate to friends when they see that their name and address is included in that certificate.

There are many uses of cryptography besides hiding data. The certificates mentioned above can be extended to limit access to services as well as software. No need for users and groups databases. No need to remember all those different passwords we each have on all the different systems we access. This could make our digital lives so much easier.

What's Holding Us Back?

We have most of the technology we need to make broad use of cryptography, but the technology is not deployed. The United States Government is actively trying to restrict access to the technology. They prevent software which uses the technology from being exported from the U.S. They also are coercing other governments to impose restrictions on the use of cryptography in those societies. The resulting fear, uncertainty, and doubt make U.S.-based businesses reluctant to develop products that use cryptographic technology, even when those products use only digital signatures, and therefore could easily get an export license.

Some companies, such as Microsoft, PGP and Sun, are actively working to have these export restrictions removed. Unfortunately Apple, as a member of the Key Recovery Alliance, is supporting the anti-cryptography stance of the U.S. Government. PGP <http://www.pgp.com/> is the original strong cryptographic technology for the masses, available throughout the world on many platforms. They have done more to relieve the U.S. export restrictions than any other company. Recently, they released a number of new Macintosh products making cryptography easy for anyone to use. They are working on a developer's kit so we developers can add cryptographic services to our applications. Microsoft is building developer-accessible cryptography into a variety of products, including a future version of Windows. Sun has developed a cryptography module (JCE, <http://www.javasoft.com/security/>) for Java 1.1, but it is not available for the Mac.

Apple's position is quite sad, especially considering that Apple holds patents to some of the strongest cryptographic technology invented and some of the least restrictive licenses to other cryptographic technology. Hopefully, Apple will soon recognize the market opportunities they could develop by making these technologies available to developers as part of the OS. Fortunately there are some Macintosh-friendly software publishers picking up some of the slack, including PGP. Consensus Development <http://www.consensus.com/> is another company that has had cryptographic technology available for years. Currently, they have a multi-platform SSL3 library, as well as other technologies in development. If you are interested in learning more about cryptography on the Mac, be sure to check out Vinnie Moscaritolo's crypto pages at <http://www.vmeng.com/mc/>. MT

MORE DEVELOPERS PROTECT.

MacHASP® PROTECTS MORE.

Mac™OS

MacHASP Packs More Into Less.

Based on state-of-the-art ASIC technology, MacHASP packs the industry's most advanced security, compatibility and flexibility into a compact and end-user-friendly dongle.

Grow With Aladdin!

The fastest growing company in the industry, with over 4 million keys sold to 20 thousand developers worldwide, Aladdin is setting the standard for software security today.

NSTL Study Rates HASP No.1!

A recent test conducted by the National Software Testing Labs†, the world's foremost independent lab, compared the flagship PC products of leading software protection vendors. The result? HASP was rated the clear overall winner - and number one in all the major comparison categories. For a full copy of the NSTL report, contact your local HASP distributor.

These days, more and more developers are choosing to protect their software against piracy. They're protecting more products, on more platforms, with better protection – and selling more as a result.

And more of these developers are protecting with MacHASP. Why? Because MacHASP offers more security, more reliability and more features than any other product on the market. Only MacHASP offers capabilities such as network support, anti-debugging envelope protection, and secure remote activation and updating.

MacHASP supports the most advanced platforms, including all versions of MacOS and Power Mac – as well as AppleTalk networks. And because Aladdin is a licensed Apple Partner, MacHASP guarantees full, transparent compatibility with the ADB standard.

To learn more about how you can protect better – and sell more – call now to order your MacHASP Developer's Kit.

The MacHASP Developer's Kit contains everything you need to protect your software today!

1-800-223-4277

w w w . a k s . c o m

ALADDIN™

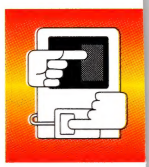
The Professional's Choice

North America	**Aladdin Knowledge Systems Inc.**	Tel: (800) 223 4277, 212-564 5678, Fax: 212-564 3377, E-mail: hasp.sales@us.aks.com
Int'l Office	**Aladdin Knowledge Systems Ltd.**	Tel: +972-3-636 2222, Fax: +972-3-537 5796, E-mail: hasp.sales@aks.com
Germany	**FAST Software Security AG**	Tel: +49 89 89 42 21-37, Fax: +49 89 89 42 21-40, E-mail: info@fast-ag.de
United Kingdom	**Aladdin Knowledge Systems UK Ltd.**	Tel: +44 1753-622266, Fax: +44 1753-622262, E-mail: sales@aldn.co.uk
Japan	**Aladdin Japan Co., Ltd.**	Tel: +81 426-60 7191, Fax: +81 426-60 7194, E-mail: sales@aladdin.co.jp
Benelux	**Aladdin Software Security Benelux B.V.**	Tel: +31 24-641 9777, Fax: +31 24-645 1981, E-mail: 100526.1356@compuserve.com

■**Aladdin Russia** 095 9230588 ■**Australia** Conlab 03 98985685 ■**Chile** Micrologica 02 7350041 ■**China** Shanghai LIRI 021 64377828 ■**Czech** Atlas 02 766085 ■**Denmark** Berendsen 039 577316 ■**Egypt** Zeineldein 02 3604632 ■**Finland** ID-Systems 0 8703520 ■**France** 1 40859885 ■**Greece** Unibrain 01 6756320 ■**Hong Kong** Hastings 02 5484629 ■**India** Solution 011 2148254 ■**Italy** Partner Data 02 26147380 ■**Korea** Dae-A 02 8484481 ■**Mexico** SiSoft 91800 55283 ■**New Zealand** Training 04 5666014 ■**Poland** Systherm 061 480273 ■**Portugal** Futurmatica 01 4116269 ■**Romania** Ro Interactive 064 140283 ■**Singapore** ITR 065 5666788 ■**South Africa** D Le Roux 011 8864704 ■**Spain** PC Hardware 03 4493193 ■**Switzerland** Opag 061 7169222 ■**Taiwan** Teco 02 5559676 ■**Turkey** Mikrobeta 0312 4670635 ■**Yugoslavia** Asys 021 623920

© Aladdin Knowledge Systems Ltd. 1985-1996 (10.96) MacHASP® is a registered trademark of Aladdin Knowledge Systems Ltd. All other product names are trademarks of their respective owners. Mac & the Mac OS logo are trademarks of Apple Computer, Inc., used under license. NSTL makes no recommendation or endorsement of any product. †The NSTL report was commissioned by Aladdin.

GETTING STARTED



by Dave Mark, ©1997, All Rights Reserved.

VerySimpleText, Version 2

Three months ago, the August Getting Started column featured a program called VerySimpleText. We built this first version of VerySimpleText using ProjectBuilder and InterfaceBuilder. We started off by editing the nib file (the first version of VerySimpleText wrapped its entire user interface into a single nib file).

We added a Format submenu to the application's default menu, thus adding a series of powerful font, text, and page manipulation features to VerySimpleText. This was done by dragging a Format menu from the menu palette in the palette window.

We also added a scrollable text area (implemented by the NSScrollView class) to the default application window. We did this by dragging a scrollable text view from the DataViews portion of the palette window. We used the NSScrollView inspector to set the autosizing for this view so the scrollable text view grew and shrank along with its containing window. We used InterfaceBuilder's Test Interface feature to test out the window, making sure it looked and behaved as we wanted it to.

Next, we added an info panel (an about box) to VerySimpleText, along with a menu item to bring up the info panel. We edited an existing menu item (Info Panel...) to create our "About VerySimpleText..." item. We used the NSMenuItem inspector to enable the item (unchecking the disabled checkbox, actually). To create the panel itself, we used the Windows portion of the palette window and dragged out our new window, changing the name of the window instance in the nib window and the window's title in the inspector. We also used the palette window to drag some default text into the new info panel.

Once the about panel was built, we created an AboutPanelController class which brought up the about panel when the "About VerySimpleText..." item was selected. Working in the Classes tab within the nib window, we first subclassed NSObject, then created one outlet (abtWindow) and one action (show:). As a reminder, think of an outlet as a variable or object you want associated with your class. When InterfaceBuilder generates the source code for this class, outlets are declared in the header file as type id. An action is a method. In this case, the show: method will bring up the about panel.

Once we were done with our nib file, we told InterfaceBuilder to generate the source files for this project and to add them to the project.

Our next step was to link the "About VerySimpleText..." menu item to the AboutPanelController so when it was selected, the show: method would get called and the panel would appear. First, we instantiated our newly created AboutWindowController class. The instance appeared in the nib window's Instances tab. We then control-dragged from the "About VerySimpleText..." menu item (it's in the menu itself) to the AboutWindowController instance in the nib window. In the inspector window, we clicked the connect button to establish this link. Now, when the "About VerySimpleText..." item is selected, the AboutWindowController's show: method will be called.

Next, we control-dragged from the AboutWindowController instance to our AboutWindow instance. When the link appeared, we moved to the inspector window and clicked on the abtWindow outlet and clicked the Connect button to establish the link. This links the AboutWindowController's abtWindow variable to the AboutWindow. We added a line to the show: method to bring up the window:

```
- (void)show:(id)sender
{
   [abtWindow makeKeyAndOrderFront:self];
}
```

The Model, View, Controller Paradigm

Before we move on to this month's additions to VerySimpleText, I thought it might be useful to talk about the Model, View, Controller paradigm, described in Discovering OpenStep: A Developer Tutorial. The Model, View, Controller paradigm is also known as MVC. MVC originated with Smalltalk-80. It categorizes objects as either models, views, or controllers.

Models are objects that emulate some process or represent some knowledge-base. For example, an Employee object represents the knowledge or data associated with an employee. It is a model of an employee. A waterworks object might model the process of converting waste water to clean water and might include the data associated with that process. In general, a model object does not have a user interface. A model object may be distributable and persistent. A model class may be reusable and portable.

View objects are the user interface of your application. Anything displayed by your application is displayed in a view. For example, a window, editable, static, or scrolling text area, button, and scroll bar are all examples of view objects. View objects have no special knowledge of the data they display. In OpenStep, the Application Kit contains a complete set of view objects, all of them designed independent of any model objects. As is evidenced by the Application Kit, view objects are reusable.

Controller objects are the mediators between model objects and view objects. Typically, you'll have one controller object per window (or, possibly, a single controller for your entire application). Your controller object communicates between a model object and its representative view object. For example, an employeeController might use data from an employee object and use that data to create a visible representation of that employee within a view object. At the application level, a controller object would take care of tasks such as loading nib files and acting as a delegate for a window or application.

Delegates

Delegates allow you to provide methods that get called by a class without actually having to subclass the class. Classes which allow delegates feature a set of delegation methods. For example, the NSWindow class features a delegation method called windowWillClose. In this month's sample program, we're going to create a class called Document which will act as an NSWindow delegate. When the NSWindow object gets ready to close, it first calls the delegate's windowWillClose method (assuming the delegate provides such a method). When we define the Document class, we'll provide a windowWillClose method so you can see how this works. You might want to take a look at the NSApplication and NSWindow classes. Their delegation methods are listed at the end of their respective files.

Loading A Nib File

As you've already seen, every application comes with at least one nib file. The nib file is similar to a Macintosh resource file, though it has much more of an object orientation. In fact, one of the primary things stored in a nib file is a set of archived objects. The information in the nib file includes information about each object (like object size and location). It also reflects the position of each object in the overall object hierarchy as well as details about connections between objects in the hierarchy (connections such as the ones we created in the August version of VerySimpleText).

An important part of the object hierarchy is the File's Owner object. **Figure 1** shows VerySimpleText's main nib file with the icon representing the File's Owner object in the upper left corner of the Instances tab. The File's Owner sits at the top of each nib file's archived object hierarchy and comes into play when you want to load a nib file other than the main nib file (which is loaded for you automatically).

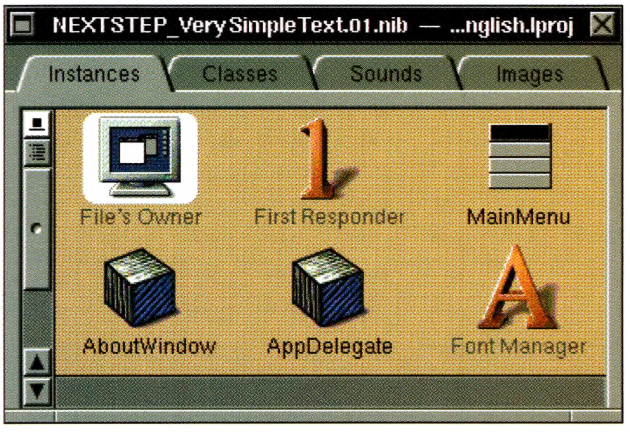


Figure 1. *VerySimpleText's main nib file, showing the File's Owner object.*

This line of code:

```
[NSBundle loadNibNamed:@"NEXTSTEP_Document" owner:self]
```

loads a nib file named "NEXTSTEP_Document.nib" and sets the File's Owner of the loaded nib file to point to the specified File's Owner. For example, in this month's sample program, we'll define a Document class and we'll tell InterfaceBuilder that the Document class will act as the File's Owner in "NEXTSTEP_Document.nib". Before the nib file can be loaded, we instantiate a Document object. In the Document's init method, we'll call the method loadNibNamed, passing in the nib file name "NEXTSTEP_Document.nib", as well as the object reference self, which refers to the Document object. This second parameter is used as the newly opened nib file's owner.

And Now, Addint to VerySimpleText

Hopefully, the quick review above brought you back up to speed on the overall structure of the August version of VerySimpleText and gave you enough background to follow this month's changes. This month, we're going to add the ability to handle multiple documents to VerySimpleText. We'll tie this functionality to the Document menu's New item.

You'll want to start off with a copy of the August version of VerySimpleText. Be sure to keep a copy of the original around

just in case. I named my original folder VerySimpleText.01 and named the copy VerySimpleText.02. Once you've made your copy, open the ProjectBuilder project in the duplicate.

- Find the file PB.project in the duplicate directory and double-click it to launch ProjectBuilder.

Next, we're going to create a new nib file.

- Click the ProjectBuilder Interfaces item, then double-click the NEXTSTEP_VerySimpleText.01.nib file.

The selected nib file will be opened in InterfaceBuilder. Now to create the new file:

- In InterfaceBuilder, select Document/New Module/New Empty.

A new, untitled nib window will appear (See **Figure 2**). If you click on the Instances tab, you'll see two instances. One is the File's Owner. If you click on the File's Owner icon, the inspector window (attributes popup) will list a set of classes and the NSObject class will be selected. We'll revisit this a bit later in the column.

Figure 2. *The new, untitled nib file.*

- Click on the Classes tab in the new nib window.
- Select NSObject.
- Select Classes/Subclass.
- Rename the new subclass from MyNSObject to Document.
- Save the new nib file.

You'll name your new nib file as NEXTSTEP_Document.nib (you can leave off the .nib if you like). Be sure to save the new nib file in the same directory as the main nib file, NEXTSTEP_VerySimpleText.01.nib (**Figure 3**).

Figure 3. *Saving the new nib file.*

- When asked, say yes to insert the file in the project.

You will now be in ProjectBuilder.

- Go back to InterfaceBuilder.
- Be sure the Document line in the nib file's Classes tab is hilited.
- Click on the outlet icon (the left of the two icons).
- Be sure that the Outlets line is highlighted.
- Select Classes/Add Outlet.
- Change the outlet name myOutlet to window.
- Click on the outlet icon to get out of outlet mode.
- Select Classes/Create Files.
- When asked whether we want to create a Document.h and .m file, click Yes.
- When asked to insert files in project, click Yes.
- We'll be back in Project Builder.

Go back to InterfaceBuilder.

- Back in the nib window, click on the Instances tab.
- Click on File's Owner.

In the inspector window, the class NSObject will be selected.

- Scroll up to Document and select it.

Document will now be highlighted when you click on File's Owner.

- Bring the original nib file to the front.
- In the Instances tab, select the MyWindow icon.
- Select Edit/Cut.
- When you are asked Do you really want to delete the window?", click Delete.

The NEW BBEdit 4.5!

It Doesn't Suck.®

still

There has never been a better time to buy BBEdit.

Upgrade from BBEdit Lite, Codewarrior, MPW, Symantec C++ or a number of other products for just $79.

This offer is available direct from Bare Bones Software, Inc. To order, call us at (617) 778-3100 or visit us online.

<http://www.barebones.com/>

MacUser AWARDS Editors' Choice 1996 Winner
Macworld 1997 World Class Award
MacUser Eddy 4½ October 1996
score card MacWEEK 5

Bare Bones Software, Inc.
P.O. Box 1048, Bedford, MA 01730 • main 617-778-3100 • fax 617-778-3111

BBEdit is a trademark of Bare Bones Software, Inc. "It Doesn't Suck" is a registered trademark of Bare Bones Software, Inc. © 1997 Bare Bones Software, Inc. All rights reserved.

- Bring the new nib file to the front.
- Select Edit/Paste.

The MyWindow icon should appear in the new nib window and the window itself should reappear.

- Hold down the control key and drag from File's Owner icon to MyWindow icon.
- When you let go, go to the inspector window and click Connect.

You've just connect MyWindow to the File Owner's outlet (in this case, the Document classes' window variable).

- In the NEXTSTEP_Document.nib window, control drag from MyWindow to File's Owner.
- Select the word delegate in the left-hand column.
- Click the Connect button.

You've just made Document MyWindow's delegate.

- Select Document/Save.

We are now done with this nib window.

- Bring the old nib file to the front.
- Click on the nextstep menu to bring it to the front.
- Select Tools/Palettes/Palettes.
- Select the leftmost palette (Menus).
- Drag a Document menu into the nextstep menu, just below Info

The new Document menu will appear, just to the right of the nextstep menu.

- In the Document menu, click on New.
- In the inspector window, select attributes from the popup menu.
- Click the Disabled checkbox so it is unchecked.
- In the Document menu, click on Close.
- In the inspector window, click the Disabled checkbox so it is unchecked.
- Click on the old nib window and select the Classes tab.
- Click on NSObject.
- Select Classes/Subclass.
- Rename subclass to AppDelegate.
- Click on action icon (on right).
- Click on Actions line, select Classes/Add Action.
- Rename new Action to new:.
- Click off the Actions icon.
- Click Classes/CreateFiles.
- Create the files (answer yes to create files and add to project).

We are now back in ProjectBuilder.

- Go back to InterfaceBuilder.
- In the old nib file's classes tab, select AppDelegate line.
- Select Classes/Instantiate.
- In the instances tab, control-drag from File's Owner t AppDelegate.
- In the inspector window, be sure delegate is selected, the click Connect.

We have just marked AppDelegate as the NSApplication delegate We won't implement any of the NSApplication delegate method in our AppDelegate code, but we could. Take a look a NSApplication and take a few of the delegation methods for a spin

- Go to the nextstep menu and control-drag from New t AppDelegate in the old nib window.
- In the inspector window, select new from the actions list then click Connect.

We've just connected the new menu item to the AppDelegate's new: method.

- Select Document/Save.

OK. That's it for the nib files. Now all we need to do is add a bit of code and we are on our way.

- Go to ProjectBuilder.
- Under Classes, select Document.m.

Here's what the code looks like now:

```
#import "Document.h"

@implementation Document

@end
```

- Edit the code so it looks like this:

```
#import "Document.h"

@implementation Document

- init
{
   //Find the nib and load it in. This instance will be the
   //File's Owner object, so we pass ourself as owner
   if (![NSBundle loadNibNamed:
            @"NEXTSTEP_Document" owner:self])
   {
      //for whatever reason, we failed. Clean up and go
      NSLog(@"Failed to load Document.nib");
      [self release];
      return nil;
   }
   return self;
}

//Since the Document is the Windows's delegate,
//it will get the following
//method called whenever the window closes.
- (void)windowWillClose:(NSNotification *)aNotification
{
   //We remove ourself as the delegate as
   //we are going to release ourselves
   [window setDelegate:nil];
   //Let garbage collection do the actual deletion
   [self autorelease];
}

@end
```

- Under Classes, select AppDelegate.m.

Here's what the code looks like now:

```
#import "AppDelegate.h"

@implementation AppDelegate

- (void)new:(id)sender
{
}

@end
```

Edit the code so it looks like this:

```
#import "AppDelegate.h"

@implementation AppDelegate

- (void)new:(id)sender
{
}

@end
```

Change it to look like this:

```
#import "AppDelegate.h"
#import "Document.h"

@implementation AppDelegate

- (void)new:(id)sender
{
   //Just instantiate a Document. It will know what to do.
   [[Document alloc] init];
}

@end
```

- Click on the hammer icon to bring up the project build window.
- Click on the hammer again to build the project.
- When prompted with the Save Modified Files dialog, click Save and build.
- Assuming the build succeeds, click on the monitor icon to bring up the launch window.
- Click on the monitor icon in the launch window to run the application.

When the application runs, select Document/New to create new windows.

Till Next Month...

Between delegates, File's Owner, and nib file loading, you've learned a lot this month. Be sure to spend some time looking at NSWindow and NSApplication to get a feel for the power of delegation. This will give you something to chew on until we have releases of Rhapsody and Rhapsody developer tools. MT

PUBLISHER'S NOTE

During the course of 1997, we've covered quite a bit of Rhapsody — especially when one takes a moment and realizes that we haven't even seen a developer release of Rhapsody yet. During this time, we've given you, the Getting Started reader, some things to start looking at. But more importantly, the industry has seen that while Rhapsody is what Apple proposes for the long term, the Mac OS is here to stay for a while — and has quite a lot of life left in it.

With that in mind, we're going to take a break from Rhapsody and Java during the next several months and review some of the most important concepts and directions in Mac OS development. If you are a more seasoned developer, you might review these articles and ponder the differences you find. If you are a novice programmer, this is a good time to really make sure you are on track. In addition, if you are practicing your development skills, you may want to look back at any one of the Getting Started articles from 1992 on — they are available on the MacTech web site and the MacTech CD.

As we finish reviewing these concepts, we will return to what is now the bleeding edge and cover things like Rhapsody and Java. Several months from now, we will know a lot more about these topics. In addition, the OS and tools will be ready for prime time.

As always, this is your magazine, so let us know what you think. Drop us a note at any of the following addresses: <letters@mactech.com>, <editorial@mactech.com>, or me personally at <publisher@mactech.com>.

—Neil

by Andrew Stone, Chief Executive Haquer, Stone

A Simple Word Processor...on Rhapsody

Here is a Simple Rich Text and Graphics Word Processor you can build yourself in 20 minutes on Rhapsody

Leveraging the power of Rhapsody is your key to building cool apps quickly. If I asked you "How many lines of code would you have to write to implement a word procesor that reads/writes rich text files (including full support for graphics of EPS, TIFF, JPEG, PICT, GIF, etc type), full font support, full rulers with tabs and hanging indents, full support for color, printing, faxing and saving as PS with embedded fonts".

But wait, before you answer, that's not all (can you hear the Ginzu knife salesman yet?)... "What if it also had ligatures, kerning, superscripting, justification, underlining, ability to drag out graphics, copy & paste of contents and copy and paste of font styles? Don't answer yet, because it will also run on Windows95 and Windows NT, and has a built-in spell checker."

If I said 13 lines of code and that you'll be done in 20 minutes, would you believe it? This is why I quit developing on the Mac in 1989 when I saw my first NeXT demo, and why I'm happy as heck to be developing on a Mac in 1997!

First look at Rhapsody

The week before Macworld Boston 97, the Rhapsody Group at Apple invited several key OpenStep developers out to Cupertino to port their wares to Rhapsody running on the PowerPC. It was a great privilege to be included in this 3 day Kitchen — a MacHack with the Apple engineers dropping in and helping us at all hours. And, as promised, create, compile, and it "just worked".

This article will take you step-by-step through the process of creating a new application, building the user interface, generating the skeleton class code, filling in the 13 lines required, compiling and testing your word processor. If you encounter terms you don't recognize, please refer to the online documentation for developers, especially /NextLibrary/Documentation/NextDev/TasksAndConcepts/DevEnvGuide/DevGuide.rtfd.

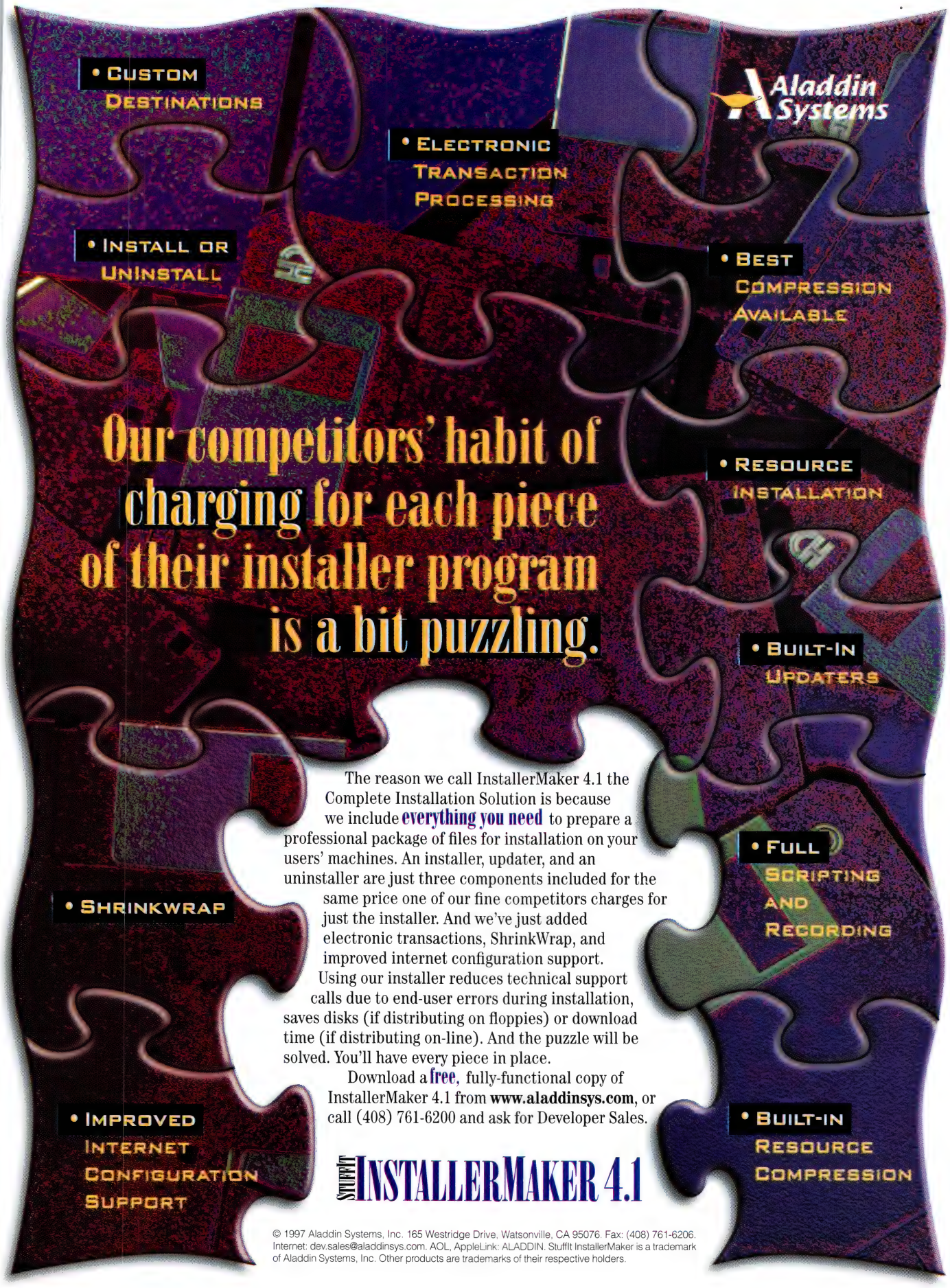
Note that this article was written before the Developer release, so be aware that the screenshots may be a bit out-of-date! Note also that the purpose is to show how easy it is to get going, not good application architecture!

How to Build It

Step 0:

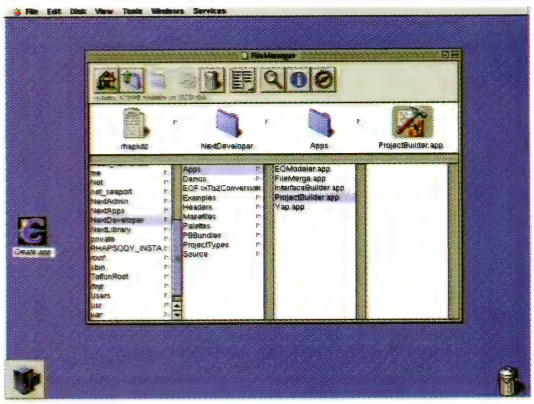
Install the Rhapsody Developer release on your power mac, if you haven't done so already. Refer to Apple's instructions on how to do this.

Andrew Stone, an early HyperTalk developer and coauthor of *"Tricks of the HyperTalk Masters"* emigrated to the NEXT community in 1989, going on to write such NeXT classics as *TextArt, Create, DataPhile* and *3Dreality*.


• Custom Destinations
• Electronic Transaction Processing
• Install or UnInstall
Aladdin Systems
• Best Compression Available
Our competitors' habit of charging for each piece of their installer program is a bit puzzling.
• Resource Installation
• Built-In Updaters
The reason we call InstallerMaker 4.1 the Complete Installation Solution is because we include everything you need to prepare a professional package of files for installation on your users' machines. An installer, updater, and an uninstaller are just three components included for the same price one of our fine competitors charges for just the installer. And we've just added electronic transactions, ShrinkWrap, and improved internet configuration support. Using our installer reduces technical support calls due to end-user errors during installation, saves disks (if distributing on floppies) or download time (if distributing on-line). And the puzzle will be solved. You'll have every piece in place.
Download a free, fully-functional copy of InstallerMaker 4.1 from www.aladdinsys.com, or call (408) 761-6200 and ask for Developer Sales.
• Full Scripting and Recording
• ShrinkWrap
• Improved Internet Configuration Support
• Built-In Resource Compression
StuffIt InstallerMaker 4.1
© 1997 Aladdin Systems, Inc. 165 Westridge Drive, Watsonville, CA 95076. Fax: (408) 761-6206. Internet: dev.sales@aladdinsys.com. AOL, AppleLink: ALADDIN. StuffIt InstallerMaker is a trademark of Aladdin Systems, Inc. Other products are trademarks of their respective holders.

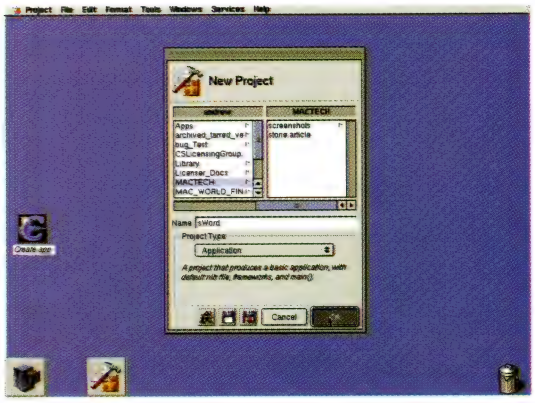
1. Launch ProjectBuilder.app (aka "PB").

   It's in /NextDeveloper/Apps, just double-click it.



2. Click Project->New...

An OpenPanel will come up — select "Application" for project type in the pop-up menu, type in "sWord" and "OK".



This will create a new directory named "sWord" with various project files:

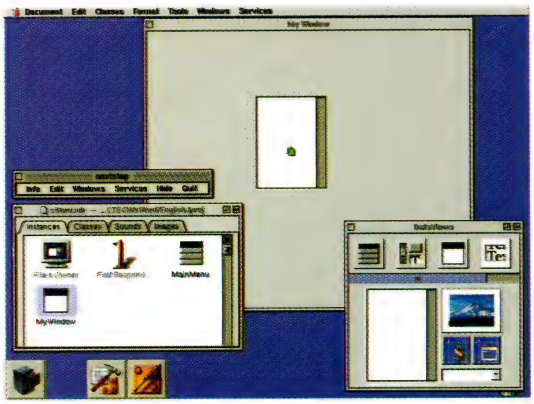
```
  PB.project: the file which maintains your makefiles.
  Makefile, Makefile.preamble, Makefile.postamble: the
makefiles.
  sWord_main.m: the source file which defines main().
  English.lproj: the directory with localized interface files
(NIBs)
  sWord.iconheader: the file which keeps track of App & Doc
icons.
```

All of these files are created and maintained by ProjectBuilder — you don't have to write a single line of code to get your app skeleton.

3. In the sWord ProjectBuilder window, click "Interfaces", and then double-click "sWord.nib" to automatically launch InterfaceBuilder (IB) — where you will design your interface, create new classes, and test your application.

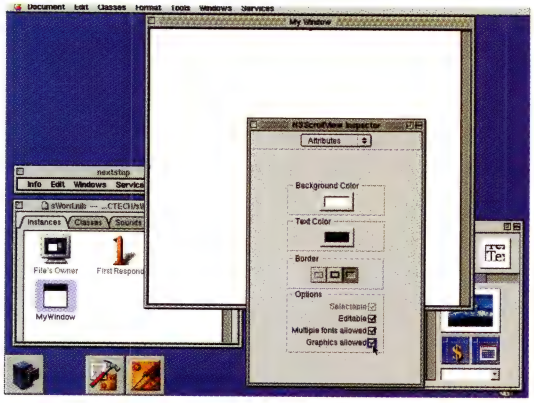
Note also the sWord-windows.nib file, which is for deploying your application on WindowNT and Window95, with no changes to your code! Since Windows organizes its menus differently, this NIB file will be different, according to Windows human interface (or lack thereof) design.

4. In InterfaceBuilder's Palette window, click the "Text" icon to load "DataViews", the palette containing Text in a ScrollView. Drag the ScrollView onto your main window, "My Window".



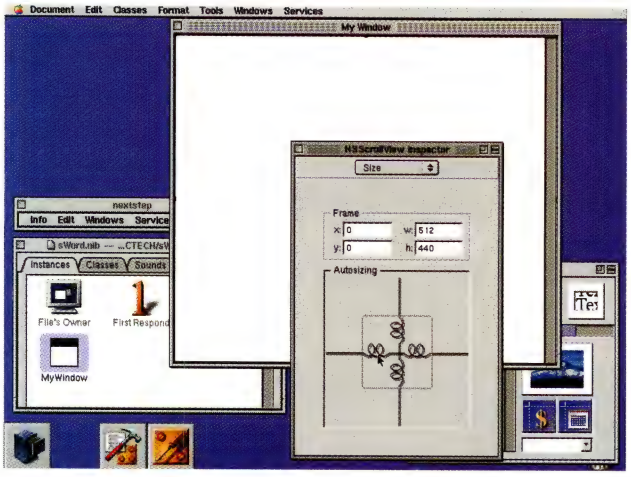
5. Move the ScrollView to the upper left hand corner of your window, and drag its bottom right knob to resize the ScrollView to fill your window.

6. We will now add the ability for the Text to be able to accept drag and drop graphics, as well as display and save them. Bring up InterfaceBuilder's Inspector Panel with Tools->Inspector. Be sure that the ScrollView is selected. All you have to do is click the "Graphics Allowed" switch!



7. InterfaceBuilder allows you to specify how objects behave when the window they are in is resized. For our ScrollView with our Text object to automatically fill the window, we must set its "autosizing". IB allows to visually set these constraints. This eliminates us having to write code to deal with window size changes.

Choose "Size" from the pop-up menu at the top of the inspector. The Size inspector allows you to specify how objects behave when a user resizes the window. Lines mean "stays fixed", springs means "size to fit". click the middle vertical and horizontal lines to allow the ScrollView to expand and contract with the window:



8. It's time to add power to our app, which we will get for free by adding various menus to our app. By default, IB gives your app some lightweight menus without the depth of functionality that is possible. For example, the stock "Edit" menu just has copy,cut,delete and paste. However, if you drag off an "Edit" menu from the IB palette, it will contain the full range of menu items and associated functionality, including the SpellChecker and a Find Menu.

PLATFORMS: APPLE SYSTEM 7 • APPLE A/UX • QNX • MOTOROLA 88OPEN • LINUX • INTERACTIVE UNIX • BANYAN • AT&T UNIX • MIPS ABI (SGI) • NT (ALPHA) • WIN NT • WINDOWS 95 • OS/2 • NLM • DOS • LYNX • SCO • SYSTEM MANAGER • SUN SPARC-SOLARIS • IBM RS/6000 • HP9000 • RS/6000 • SUN O/S 4.X •

Java Interface! New c-tree 6.7! Enhanced Servers! Check it out!! www.faircom.com

Q: What does it take to ... superior client/server ...
A: A SUPERIOR SE...

START with the most advanced client-side SDK on the market: c-tree® Plus at $895.

- **Complete "C" Source code**
- ROYALTY FREE (Client Side)
- Multiple supported protocols
- Fast, portable, reliable
- Powerful features like transaction processing
- Win95, NT, and Windows 3.1 ready

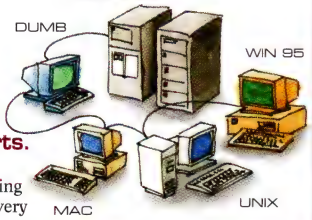
RESULT
A solid, economical, easily deployable product that fits your needs.

- Portable
- Scalable
- Exceptional Performance
- Flexible
- Easy Server distribution
- Convenient OEM terms

ADD a strong, multi-platform, industrial-strength Server that supports.

- File mirroring
- Heterogeneous networking
- Automatic disaster recovery
- Multi-threaded design
- Best price/performance available: from $445- $3745

FAIRCOM® Server



Heterogeneous TCP/IP Network

You can't find a better client SDK with these features!
Over sixteen years of proven reliability and performance.
No one else supports over 30 platforms in this price range!

c-tree Plus®
- Complete C Source
- Single/Multi User
- Client/Server (optional)
- Full ISAM functionality
- No Royalties
- Transaction Processing
- Fixed/Variable Length Records
- High Speed Data/Index Caching
- Batch Operations
- File Mirroring
- Multiple Contexts
- Unsurpassed Portability

FairCom® Server
- Client/Server Model
- Transaction Processing
- Requires <2MB RAM
- Online Backup
- Disaster Recovery
- Rollback - Forward
- Anti-Deadlock Resolution
- Client-side "C" Source
- Multi-threading
- Heterogeneous networking
- File Mirroring
- OEM/Source Available

FOR YOUR NEXT PROJECT CALL FAIRCOM: YOU CAN'T FIND A BETTER HETEROGENEOUS CLIENT/SERVER SOLUTION!

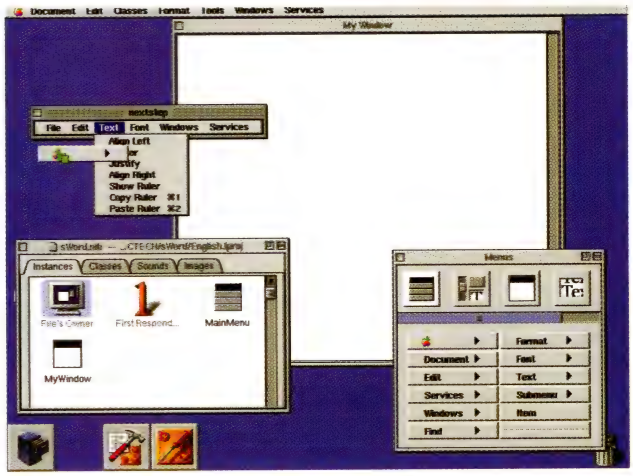
Also inquire about these FairCom products:
d-tree™
r-tree®
ODBC Driver

FAIRCOM® CORPORATION
Since 1979
WWWeb Address: http://www.faircom.com/
800-234-8180
U.S.A. phone (573) 445-6833 fax (573) 445-9698
EUROPE phone (035) 773-464 fax (035) 773-806
JAPAN phone (0592) 29-7504 fax (0592) 24-9723

First, delete some of the stock ones provided by clicking the menu item, and choosing Edit->Cut. (This may change for Rhapsody Developer Release.) Your menu should look something like this now:

9. In IB's Palette window, click the Menu icon to load the Menus. Drag over the following menus from the Palette to the sWord Menu window:

```
Apple
Document (rename this to "File")
Edit (replace the other one - this one has a Spell Checker
in it!)
Font
Text
```

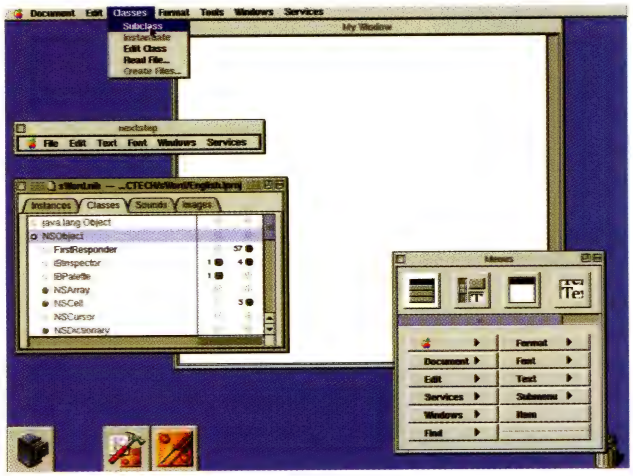


Click the File menu to drop it down. From the IB Palette window, click and drag from the "Item" button to the sWord File menu. Rename the new menu item to "Page Setup...". Drag over another menu item, and rename this one to "Print...". These items may be there automatically in the Rhapsody Developer Release.

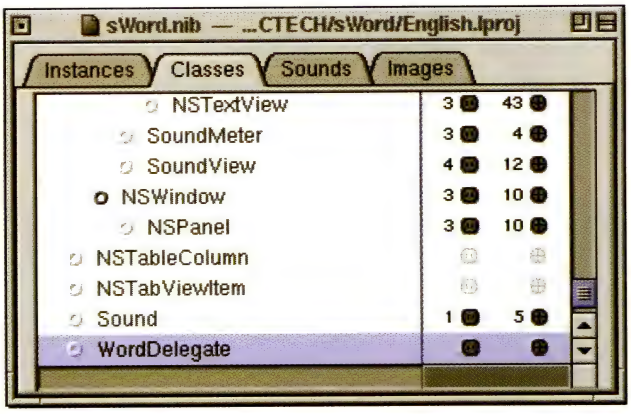
10. Now we are going to design a simple object, add its outlets (technically speaking, "instance variables") and actions (Objective C methods), and have InterfaceBuilder automatically create the source files for this new class. All you will have to do is add the few lines of code reproduced below to make these custom actions do something useful.

Click the "Classes" tab in the window with the "Instances" of objects in your interface, and an outline of the class hierarchy will be displayed.

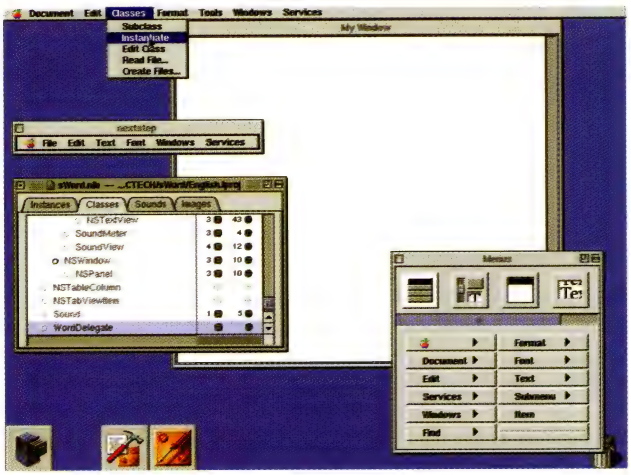
Click "NSObject".
Select "Classes->Subclass" from the menu bar.



Rename "MyNSObject" to "WordDelegate".



11. We will now add an instance of this new class, WordDelegate to our app. Choose "Classes->Instantiate".



The industry's solution showcase

MACWORLD Expo Conference & Exhibits: January 6-9, 1998

You depend on the Macintosh to get work done faster, to stay connected with colleagues and friends, and turn your best ideas into action. Which is why you can't afford to miss the largest event that brings you the world of the Mac.

Hundreds of New Products and Industry Insights!

Only at MACWORLD Expo can you see and try thousands of products first-hand... learn from Macintosh experts through conferences and keynotes... talk with other Macintosh users and vendors... and stay on top of new developments that could impact your buying decisions. You'll gain valuable insights into how innovative companies are unleashing the power of the Macintosh OS.

Come evaluate cost-saving solutions for:

- Publishing, entertainment and multimedia
- Web site design and Internet navigation
- Networking, intranets and enterprise-wide connectivity
- Education and R&D
- Business and telecommuting

Make plans to attend MACWORLD Expo/San Francisco today!

If the Macintosh is part of your business, MACWORLD Expo is your lifeline. MACWORLD Expo/San Francisco is the industry's premier Macintosh event, and dramatic software introductions scheduled for 1998 may make this year's event the most exciting ever. Gain new insights into the future world of Macintosh... see the hottest and coolest new products... and get the inside view of the Macintosh OS platform. Register to attend today.

Owned & Produced by:
IDG EXPO MANAGEMENT COMPANY

Sponsored by:
Macworld MacWEEK

For Details:
www.macworldexpo.com
or 800.645.EXPO

Macworld EXPO

san francisco

NEW! **Macworld/Pro Conference**

The Macintosh Professionals Conference only at MACWORLD Expo/San Francisco, January 5-7.

Please send more information on MACWORLD Expo

❑ **San Francisco/Jan. 1998**

❑ **Attending** ❑ **Exhibiting** **MT**

Name _____

Title _____

Company _____

Address _____

City/State/Zip _____

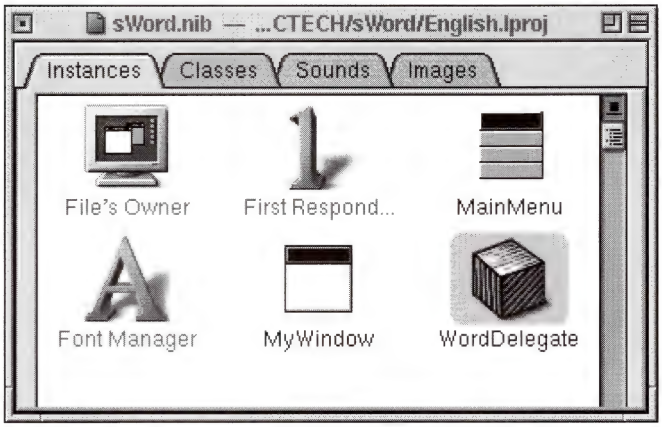
Phone _____ Fax _____

email _____

Mail to: MHA Event Management, 1400 Providence Highway, P.O. Box 9127, Norwood, MA 02062. Or Fax to: 781-440-0363

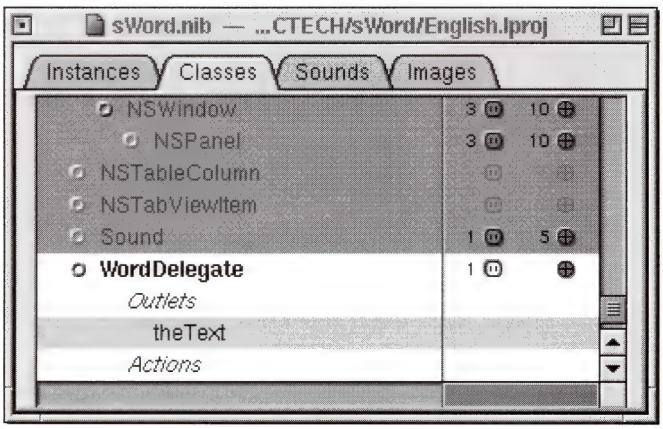
THIS IS NOT A REGISTRATION FORM.

Click the "Instances" tab to reveal what we've added:

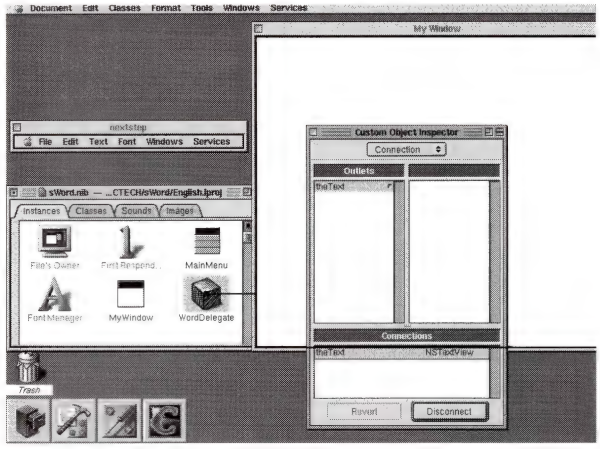


12. Now, return to the "Classes" tab to add the actions (the objC method called when you click a menu item or button) and outlets (the instances of objects that your WordDelegate knows about) to the WordDelegate Class.

13. Click the "outlet plug" icon to the right of the WordDelegate, and click "Outlets" when it appears. Hit the <RETURN> key to create a new outlet, "myOutlet". Double-click to select the text and rename this outlet to "theText". This outlet will become an instance variable in our new WordDelegate class, so we can refer to the NSTextView object programmatically and send messages to it, but we'll "hook it up" in InterfaceBuilder. "Hooking Up" is the visual programming equivalent of assigning both an action and a target for menu items, buttons, controls, etc.

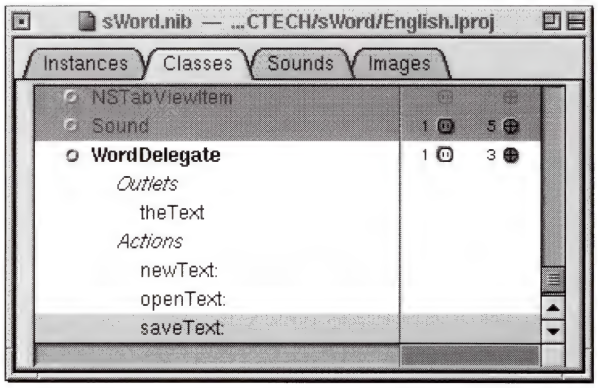


14. Connect the WordDelegate's outlet "theText" to the NSTextView which is inside of the ScrollView by control-dragging from the WordDelegate instance. Double-click "theText" in the Inspector's Outlets Browser.



15. Now, we'll add the functionality to our WordDelegate by adding the actions to which it can respond.

    a. Click the small cross icon on the right to drop down the "Actions".
    b.Select Actions, and type <RETURN> to add a new action.
    c. Rename it "newText:".
    d. Again, type <RETURN> and rename the new action to "openText:".
    e. Again, for "saveText:".



16. Now, we'll connect our menu items to the object which performs the action and select the action to be performed.

Click the File menu to drop it down. Hold down the Control-Key, and click-drag from the Open... menu to the instance of WordDelegate, and release the mouse. Black lines will connect them up. Select "openText:" in the Actions browser, and click "Connect" in the NSMenuItem Inspector (double-click openText: to avoid this second step).

The number one AppleScript environment

widens the lead.

**For professionals,
for novices,
for webmasters,
for solutions providers:**

There's only one serious choice.

**Always the most powerful,
the most feature-packed,
and at the same time,
the easiest to use.**

And it just got better.

It's already the only tool that lets you watch local variables, trace variables, or change values and fix problems while stepping.

Now, Scripter hits version 2.0,

with still more unique features:

Handler debugging: Debug handlers without modifying your scripts.

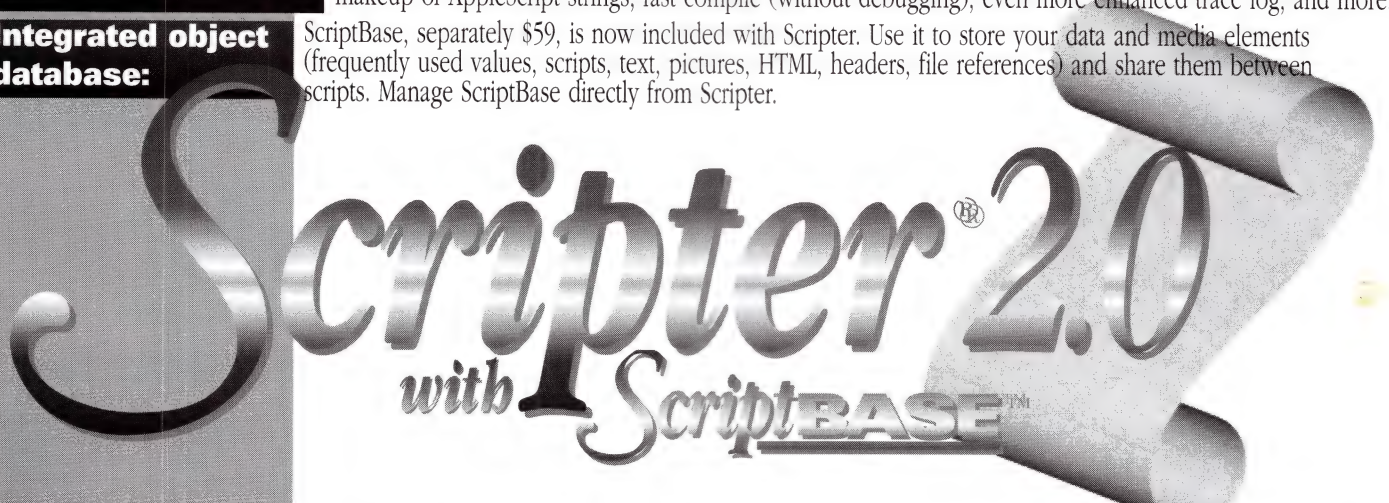
Applet and CGI simulation: Interactively debug live interapplication messages sent to a script application or AppleScript CGI from an applet, your Web server or FaceSpan 2.x. You won't have to modify your Web pages or your CGIs.

Live editing: Debug script applications and CGIs in their normal environment.

Object Map: Explore a graphical view of the class hierarchy of scriptable applications.

Plus: Construct generators; associated terminology; search backwards; faster searching; more navigation tools; makeup of AppleScript strings; fast compile (without debugging); even more enhanced trace log; and more.

Integrated object database: ScriptBase, separately $59, is now included with Scripter. Use it to store your data and media elements (frequently used values, scripts, text, pictures, HTML, headers, file references) and share them between scripts. Manage ScriptBase directly from Scripter.



Buy online at
www.mainevent.com
or order, toll-free
1 800 616 8320
info
+1 202 298 9595
email
info@mainevent.com

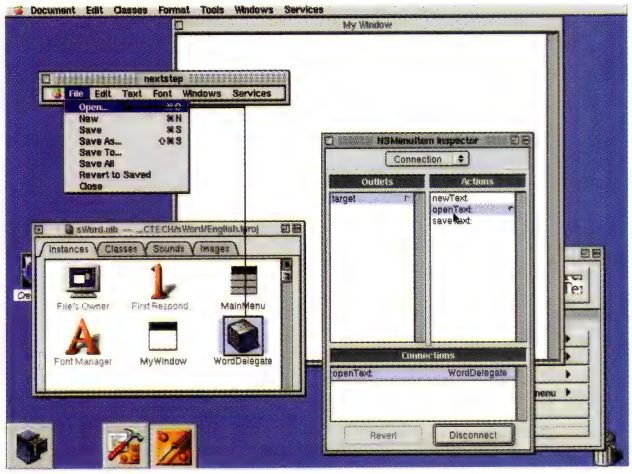
If you're an AppleScript novice, Scripter will take you by the hand and show you the correct syntax for your statements. If you know what you're doing, Scripter will help you do it faster than any other editor. And Scripter is the only tool on the market with *interactive debugging:* You catch the offending code in the act. Only Scripter lets you fix the problem and continue debugging!

We designed Scripter for productivity and user-friendliness. People come to us for Scripter's industrial-strength debugger, but what really impresses them is the speed and ease at which Scripter allows them to work. It's been used to build major corporate process automation systems.

Scripter offers power-assisted statement construction. Vocabulary access in a single mouseclick. Multifunction find and replace. A variable watcher and expression evaluator. Tools to change variable values or try out commands, in context, in the middle of debugging. Automatic navigation to subroutines. Background processing. An enhanced trace log. Extensive Drag and Drop support. We could go on, but we just don't have the space here. So get a copy today, and start scripting with power!

Design: Boris Levitin

Scripter is a registered trademark, and ScriptBase is a trademark, of Main Event Software, Inc. FaceSpan is a trademark of Digital Technology International. Macintosh and AppleScript are registered trademarks, and Mac is a trademark, of Apple Computer, Inc.

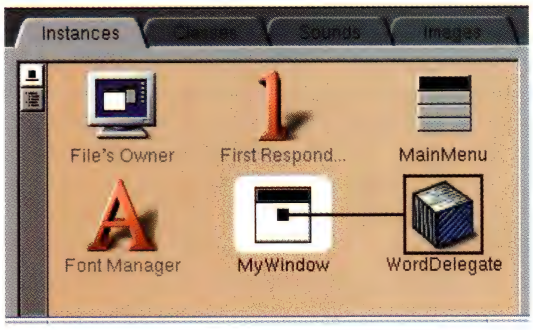


Repeat this for "Save" and "New", but double-click the actions "saveText:" and "newText:" respectively.

17. In the same control-drag manner, connect the Page Setup... to "First Responder" object in the Instance Browser. This is a very cool "placeholder" object which will send the method to the most appropriate object for the current context of the application. In Rhapsody, there is this notion of a responder chain which begins with the active user interface object (such as the Text if your cursor is blinking there), the window's delegate, the window, then the Application's delegate, and finally, the Application itself. The action is sent to the first object in the chain that responds to it (ie First Responder), and if none do, the menu item is automatically dimmed and disabled. For a full description of the Responder chain, you can access the online documentation via ProjectBuilder: click "Frameworks" -> "AppKit.framework" -> "Documentation" -> "Reference" -> "Classes" -> "NSResponder.rtf". You will quickly learn how useful these docs are!

For our WordDelegate object to get these First Responder method calls, we'll must insert our WordDelegate into the First Responder chain.

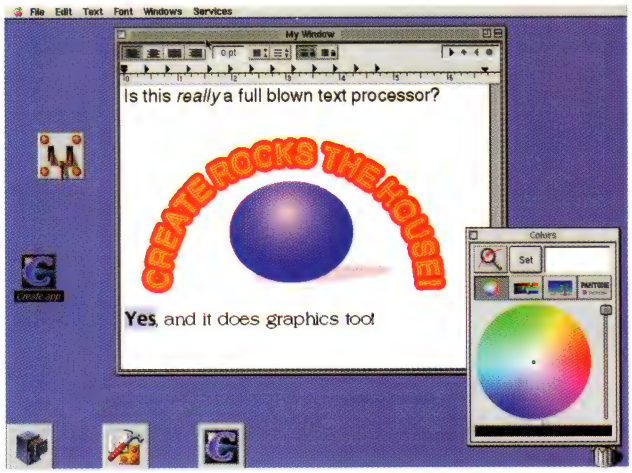
Control-Drag from the "My window" icon in the Instance Browser to the "WordDelegate" instance, and select "delegate" outlet in the Outlets browser of the Window Inspector.



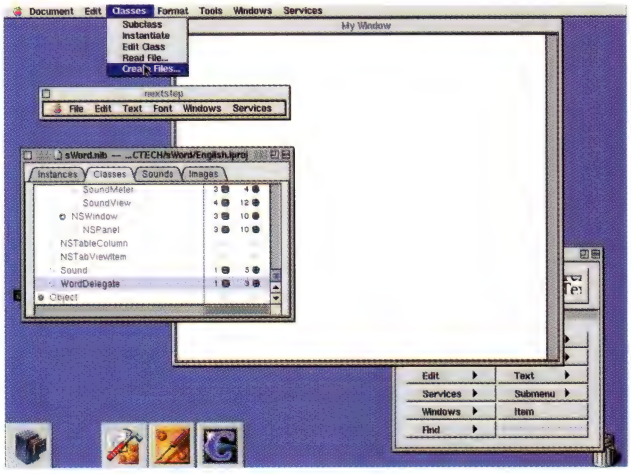
18. Control drag from "Page Setup..." menu item in the dropped down File menu to the First Responder icon in the Instances browser. Double-click "runPageLayout:" in the Inspector's Actions browser.

Likewise, Control-drag from "Print..." menu item to the Text portion of the ScrollView. Double-click "print:" in the Actions browser.

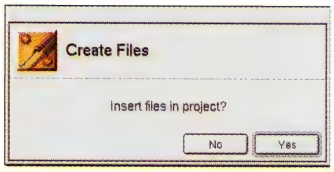
19. Now, let's test drive our app within InterfaceBuilder by choosing "Document->Test Interface". This then "runs" our application in an interpreted environment, so you can try out typing text, changing fonts, bringing up the ruler, dragging in graphics and so on. You won't be able to save or open yet, because we need to write that code, compile it, and run the compiled version to see additional functionality over what is already part of the runtime system.



20. We've designed an object, hooked it up, now let's ask IB to make the skeletal source files. Click the "Classes" tab and select the WordDelegate Class. Choose "Classes->Create Files..." from the menu bar.



After verifying that creating classes is what you want to do, InterfaceBuilder will then ask you if you want insert these new files into your sWord project.



Click "OK", and then ProjectBuilder will come up showing you the new source files.

21. Click "WordDelegate.m" under the Classes category in ProjectBuilder. The skeletal source file will be displayed, now it's time to write those 13 lines of code, and you'll see the beauty and elegance of Rhapsody!

22. Type in this code, I added the comments for your edification, so they don't count in the number of lines of code!

```
***** WordDelegte.m *****

#import "WordDelegate.h"
@implementation WordDelegate
- (void)newText:(id)sender
{
// empty out the text with the empty NSString:
    [theText setString:@""];

// Set the window's title to be untitled:
    [[theText window]setTitle:@"Untitled"];
// bring the window up in case the user has closed it:
    [[theText window] makeKeyAndOrderFront:self];
}

- (void)openText:(id)sender
{
// Get a new Open Panel
    NSOpenPanel *openPanel = [NSOpenPanel openPanel];

// Have it run modal and look for files of "rtf" or "rtfd" type:
  if ([openPanel runModalForTypes:[NSArray
arrayWithObjects:@"rtf",@"rtfd",NULL]]) {
// we have a valid file, ask theText to read it in
        [theText readRTFDFromFile:[openPanel filename]];
// Update the window's name with the filename, but in a readable way:
        [[theText
window]setTitleWithRepresentedFilename:[openPanel filename]];
// bring the window up in case the user has closed it:
        [[theText window] makeKeyAndOrderFront:self];
  }
}

- (void)saveText:(id)sender
{
// Get a new Save Panel:
    NSSavePanel *savePanel = [NSSavePanel savePanel];

// Set it to save "rtfd" files:
    [savePanel setRequiredFileType:@"rtfd"];

// Run modal, which returns YES if a valid path is chosen:
  if ([savePanel runModal]) {

// Ask the text to write itself to the chosen filename
// But don't make it back up before
// Set atomically:YES if you want "save backups", slower but more secure
        [theText writeRTFDToFile:[savePanel filename]
atomically:NO];

// Update the title bar of the window
        [[theText
window]setTitleWithRepresentedFilename:[savePanel filename]];
    }
}


@end
************************
```

23. Click PB's Build icon, which brings up the "sWord — Project Build" panel. Click the Hammer icon again, and your app will get compiled. You can run it from within PB, or simply double-click the sWord.app in your sWord directory.

24. To build an installed version, click the "Options" panel button, and choose "Install" for the make target, then select the architectures you wish your app to run on. Build again. This will install the sWord.app into your ~/Apps directory, after "stripping" it to its smallest possible size.

23. Launch sWord.app and try it out!

Epilogue

That was easy, eh? Here are some things you can do to enhance your word processor:

0. Rename "My Window" to "Untitled" so that the title starts in the right state. This is trivial to do in IB's Inspector, "Attributes" when the window is selected in the Instance browser.
1. Add multiple documents to your app by creating a separate nib file which is owned by the WordDelegate class. See /NextDeveloper/Examples/AppKit/TextEdit for a very powerful, yet simple TextEditor which allows multiple docs (Document.h & Document.m).
2. Add an Application Icon by creating a 48*48 icon (/NextDeveloper/Apps/IconBuilder.app), saving it, and then dragging it from the FileViewer to the "Project" icon well in ProjectBuilder's inspector, and recompiling.
3. Add Find — TextFinder.h, TextFinder.m and FindPanel.nib and FindPanel.strings from TextEdit contain the functionality you need. This is much vaunted "code reuse" of object programming!
4. Create methods for SaveAs... Again, look at the document architecture in the /NextDeveloper/Examples/Appkit folder.
5. Add an "About..." panel. Drag in a panel from the IB palette and connect the "About..." menu item to this panel, with an action of "orderFront:".
6. Add Tool Tips. Simply create an rtf file for each object that you want give popup help to, and attach to the user interface object in IB, Inspector->Help.
7. Make the OpenPanel and SavePanel remember their last opened directory by making those variables static, and "retaining" them.

```
- (void)saveText:(id)sender
{
// Create a static variable lives between invocations:
    static NSSavePanel *savePanel = nil;
// If it's the first time through, get a new Save Panel:
    if (savePanel == nil) {
     NSSavePanel *savePanel = [[NSSavePanel savePanel]
retain];
    }
// Now, it will 'remember' it's last chosen directory...
```

Anyway, I hope this gives you a taste for the elegance and comfort of finely integrated Rhapsody development tools. MT

by Evan Trent

Designing a Point of Sales Application

Finding the hardware and writing the software to set up a Macintosh as a cash register

A Brief Introduction

"If this thing beeps again, I want you to throw it out the window," I remember one of my mother's employees murmuring through his teeth. This was hardly the first indication that the cash register system we were using was eventually going to be replaced. Yet, I really hadn't expected that I would be responsible for creating the new system.

A Unique Solution

I realized that several of the flaws inherent in a terminal-based cash register are related to its such poor user interface. On other side of that spectrum lies the Macintosh, providing a graphical user interface which enables even the novice user to perform a variety of tasks with great ease. The Mac is the ultimate "idiot proof" machine, that was exactly what we needed: a system that could provide a simple user interface but wouldn't let the user do something illogical. As I thought more about it, I came to the conclusion that I could write a point-of-sales (POS) application for the Macintosh that would provide all of the features we needed and at the same time make use of the Macintosh GUI. It was to be a match made in custom application heaven.

A Journey Into POS

While the software I wrote is an essential element in the point-of-sales system we currently use, MacTech readers certainly have little difficulty writing their own software. What is arguably of more importance is the hardware, and I intend to discuss the issues I encountered in dealing with the hardware in great depth. However, to better understand how the hardware functions, it makes sense to discuss the software design. As is the case with many applications, but especially in the case of a POS application, of foremost importance is the user interface.

Designing the User Interface

While the user interface is fairly simple, it is exactly this simplicity which makes it so flexible. We had previously been using a programmable keyboard with a Panasonic 5000 cash register system. You've probably seen similar systems in supermarkets or cafeterias. To charge a customer for an item, the employee presses the button on the keyboard corresponding to that item. Nearly all such systems then proceed by beeping loudly for some unknown reason. (The customer doesn't think much of this, but the employee is either going deaf, or insane, or both.) To prevent an awkward transition from occurring between our old system and the new Macintosh based system, I decided to place the programmable keyboard in a window on the Macintosh screen (**Figure 1**). This may seem odd, but it works very well for a variety of reasons.

Evan Trent <evan@sover.net> is the 18 year old son of Alice and Walter Trent, owners of Alice's Bakery and Café in Norwich, VT. He has been programming as a hobby for several years, and recently wrote a Macintosh-based point-of-sales application for his parents' business. He is currently a senior at Governor Dummer Academy, the nation's oldest boarding school, located in Byfield, MA.

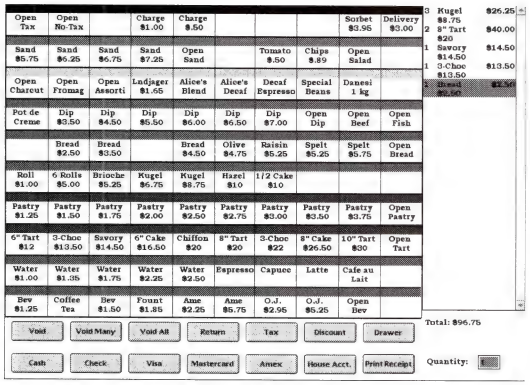


Figure 1. *The on-screen programmable keyboard.*

The biggest advantage over the old "physical" programmable keyboard approach is the increased ease of changing item attributes. No longer does the employee have to reprint a keyboard template and tape it down. (This proved to be absolutely impossible on the old system, because the little boxes on the template never properly overlapped the buttons on the keyboard; don't ask me why.) The "virtual" (on-screen) programmable keyboard is comprised of 100 square buttons, arranged in a 10x10 grid. Each row is color coded to ease strain on the eyes when searching for a particular item. Typically items are placed in rows with other items in their respective, colored subgroups (sandwiches on one row, beverages on another) and thus the employee associates each type of item with a color. This reduces the time it takes to find any given item on the screen.

To the right of the on screen keyboard, there is a scrollable list. This list provides a running queue of items in the pending sale. The first column contains the quantity of the item, the second column contains the name of the item, and the third column contains the net price of the item. This list proved to be the most applauded enhancement over the old system.

Underneath the programmable keyboard there are two rows of buttons. The first button, labeled "Void", removes an item from the queue, or if there is a quantity greater than one of that item, it decreases the quantity by one. "Void Many" removes the item from the queue regardless of quantity. "Void All" removes all items from the queue. "Return", which is intended to be used when a customer brings back damaged or lame goods, will negate the price of the selected item. "Tax" will tax an item, or "untax" an item if it already has been taxed. "Drawer" simply opens the cash drawer. "Print Receipt" will print a receipt for the last sale, even if it has already been processed. The user can define which payment methods automatically print receipts and which don't. For example, most customers don't want a receipt for cash or check transactions, but do for credit cards. If the customer does want a receipt for a cash or check transaction, the user simply can press the Print Receipt button. This saves a lot of paper.

Underneath the queue of items, there is a running total, and underneath that is an edit text box labeled "Quantity". This text field will only accept numbers, and will set the quantity for the next item to that number it contains. For example, to add 99 Espressos to the queue, the user would type 99 (which would be entered into the edit text field) and then press the Espresso button. After an item is added to the queue the quantity text field is automatically reset to 1 and the text is selected such that any number typed will replace the 1. This proved to be an efficient method of entering multiple items with a minimum of fuss.

Form Follows Function

There were several other employee interaction problems with the previous system, but the most serious design flaws occurred within the managerial functions of the cash register. To change an item's attributes, the employee had to retrieve the "manager's key" and turn the register into "manager" mode. This also only could be done from one register (the master register, as opposed to the other slave registers). From this point on the English language ceased to exist. There were code numbers for everything. Why is an employee (or manager) supposed to know that to add meals tax to an item, he must set "Special Flag 2" to 0010? This was completely absurd. Our employees invest their time in learning about food, not special code numbers. When items needed to be changed, I was the only person who could change them, and it took forever because the items had to be changed one at a time.

I disposed of the old approach to editing item attributes and decided upon a better method. Because the employees were already familiar with the on screen keyboard, why not have that be preserved throughout the editing mode? When the employee selects "Change Item Attributes" from the "Register Functions" menu, a new dialog box appears (**Figure 2**) presenting the on screen keyboard.

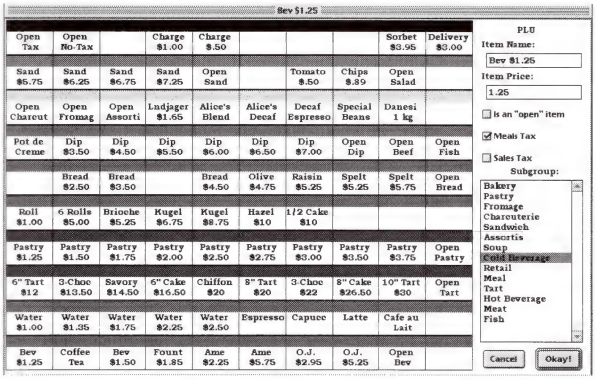


Figure 2. *A similar interface is used for editing item attributes.*

Where the item list used to be, there is a new set of dialog items accepting input for the item's name, price, subgroup and other options. Three checkboxes are visible. The first toggles the "open" status of an item. If an item is "open," it does not have an assigned price, rather it asks for a price every time it is added to the queue. An example of an "open" item is a cheese which is weighed on the scale. After the scale prints a label, the user presses "Open Fromage" and types in the price which appears on the label. If the "open" checkbox is checked, no item price may be entered. Likewise if an item has a price and the user checks the "open" checkbox, the price is removed. The old system

allowed a price to exist even if an item was an "open" item. This created a lot of confusion. Additionally, there are two checkboxes, one for sales tax and one for meals tax. While an item can be taxed at any time, certain items always carry a tax with them (sandwiches for example) and thus it saves time to pre-tax them. Any item which has the meals or sales tax checkbox checked will automatically add the appropriate tax to its price.

Changing an Item's Attributes

Changing an item's attributes is a trivial task. The employee clicks an item's button, and then edits the content of each of the dialog items containing the appropriate attributes. He can then click another item's button to go on changing that items' attributes, or simply click the Okay button to save changes for that one item (or Cancel to revert to the previous attributes). This allows items to be edited very quickly, and in a batch manner. The user will always know what item he is editing because the name of the item is displayed in the window's title bar. Thus if a user changes an item's name and forgets which item he is editing, he can glance up at the title bar. This proved to be very well received.

Let Me Speak to the Manager

Additional managerial functions were improved. Tracking, polling and report generation were preserved but were executed in more logical manners. The production staff wanted to check on how many sandwiches had been sold, for example, at 3:00 PM without closing out for the day. The old system required a key and a slew of codes to acquire this information. The new system made this simple. Closing out for the day on the old system took an eternity because the system was printing all kinds of ridiculous reports that were uninformative and essentially useless. About half of the reports generated were of use to us. Those were the reports I carried over to the new system. Additionally, new reports were added to the system. One such report was the price range report, which breaks down the number of sales by total. For example, it might inform us that ten sales were between $0 and $5 and fifty were between $5 and $10. Other reports provide information on totals by item, subgroup, tax, and hour of the day (as well as day of the month when closing out a month). The new system takes about ten seconds to close (probably because it's PowerPC native) and for each day saves a report in a file named after the date. The old system printed one report, and that was it. You couldn't print it ever again.

Put That On My Tab

A feature we added from scratch was house accounts. We wanted the ability to run up a tab for each of our regular customers, keeping track of what they had purchased. Here we used the List Manager to list accounts in alphabetical order by last name. (**Figure 3.**)

Mac, go out and touch the world for only $199!

ADB I/O

ADB I/O lets your customers' Macs control things, it lets them feel.
ADB I/O lets the Mac be part of the physical world.

Thousands of Uses

Science, Multimedia, Children's Museums, Home Automation, Theatre Stages, Industrial Testing, Medical Research, Bonsai Watering, Robotics, Weather Stations—anything that can be electronically measured or controlled can use the ADB I/O.

No Serial Ports Occupied

ADB I/O uses the Apple Desktop Bus to communicate inputs and outputs to and from your Macintosh. (Maximum polling frequency is 90 Hz.) No external power supply is needed.

Eight I/O Channels Provided

Four relays for output. Four channels for Digital In, Digital Out or 8-bit Analog In.

Extensive Software Support

With ADB I/O and nearly any environment,* it is easy to build customized applications for your control and data acquisition needs.
For more info, visit us at **www.bzzzzzz.com.**

BEEHIVE TECHNOLOGIES INC.
The High-Tech Cottage Industry Cooperative On the Internet

*ADB I/O supports three language environments, two scripting environments, multimedia development environments, and other specific application environments with more on the way. Visit our home page to find out more and to download the complete manual as well as all supporting software at: http://www.bzzzzzz.com.

(818) 304-0664 voice. e-mail: contact@mail.bzzzzzz.com (818) 568-1530 fax. © 1997 BeeHive Technologies, Inc. All rights reserved.

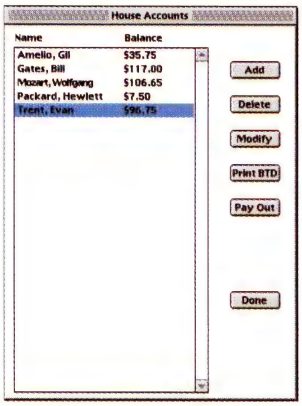


Figure 3. *This dialog lists house accounts in alphabetical order.*

From the dialog shown in **Figure 3**, the user can "Add" a new account, or "Delete" or "Modify" an existing account. Of course, the user cannot delete an account with an outstanding balance. Clicking the "Print BTD" button will print a "Balance to Date" report, which contains a list of every item purchased in reverse chronological order since the balance was last paid. Thus, a customer can at any time ask for their balance and will receive a full statement. Clicking the "Pay Out" button will print two copies of that statement, with the words "Balance Paid in Full" at the bottom. This serves as the customer's proof of payment. The account balance is reset to $0.00 and the drawer opens to accept payment. Clicking the "Done" button simply dismisses the dialog.

Modifying an Account

Additional information (such as address and phone number) is added when an account is created, and can be modified at any time. (**Figure 4.**)

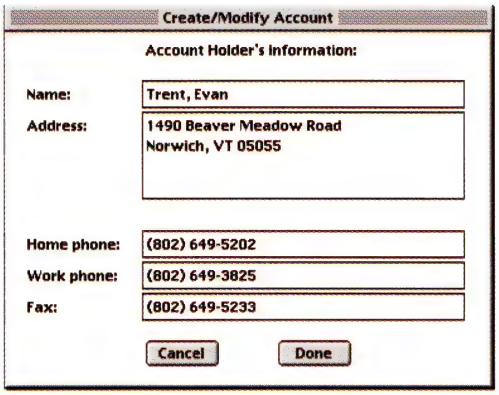


Figure 4. *An account holder's personal information may be altered through this dialog.*

Build great applications... Better Cheaper Faster!

Great | Good'nuf

Tools Plus

LIBRARIES + FRAMEWORK

Make cool apps while others are still reading their manuals.
Professionally polished. Wickedly fast. Delightfully efficient.

"...the routines are more compact and faster than anything you might write...Every element of Tools Plus is useful... a bargain compared with coding these routines yourself."

MW★★★★★ Macworld

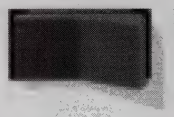
REC PLAY

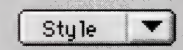
You build the interface.
Tools Plus provides the infrastructure.
It makes all your pieces work together as an application.
With only a few hundred routines, Tools Plus thins your code by tens of thousands of lines. You see results sooner.
Changes are a snap.

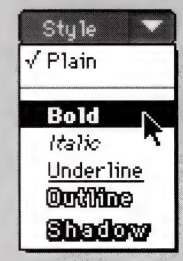
"All in all, it's an incredibly rich collection of tools... If you are interested in developing applications that have 'quality' written all over them, then Tools Plus is for you." **MacTech MAGAZINE**

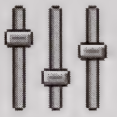
Yes, Tools Plus has it!

- **Create any element** using a single routine
- **Everything works** as soon as you create it
- **Automates** all standard GUI elements
- **Windows**, **Tool bar** and **floating palettes**
- **Buttons** (all kinds, flat and 3D)
- **Scroll bars** (speed control, live scrolling)
- World-class **custom controls**
- **Fields** (w/scroll bars, filters, auto-editing)
- **Picture buttons** (the best anywhere)
- **List boxes**
- **CDEF** and **LDEF** automation
- **Cursors** (color, animated, auto-change)
- **Menus** (pull-down, hierarchical, pop-up)
- **Edit menu** (undo/redo, automatic editing)
- **Panels** (3D, group boxes, lines, panes)
- **3D titles** (raised or inset)
- Extended multi-monitor and color support
- **Clipboard** automation
- **Dynamic alerts** (no resources required)
- **Event processing** automation
- Over 500K of custom fonts, icons, cursors, and other useful resources

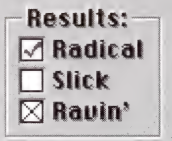


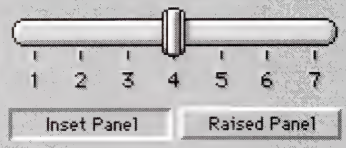






Plus thousands more exciting features and services!





FREE

SuperCDEFs™ $89 value
✔professionally designed and crafted controls
✔dozens of 3D & flat buttons, tabs, sliders

Tools Plus for

Symantec (THINK) C/C++ *(68K)*	$149
THINK Pascal *(68K)*	$149
THINK C/C++ & THINK Pascal	$199
CodeWarrior Bronze *(68K)*	$199
CodeWarrior Gold *(68K and PowerPC)*	$249

(We accept VISA and Amex only. Add $10 for shipping.)
*Call for Academic pricing

Water's Edge Software
2441 Lakeshore Road West, Box 70022,
Oakville, Ontario Canada L6L 6M9

Orders and Enquiries:
Phone: (416)219-5628
Fax: (905) 847-1638

WaterEdgSW@aol.com

Free Evaluation Kit:
Available at our web site
http://www.interlog.com/~wateredg

Water's Edge Software

"House Account" is considered a payment method just like cash or check except that obviously the money is not in the drawer at the end of the day. When a transaction is made, if the employee chooses house charge as the method of payment, he must pick from a list of house accounts (**Figure 5**), or he can create a new one at that time.

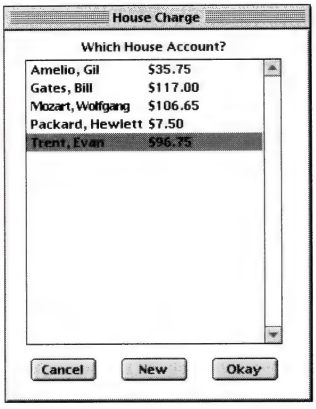


Figure 5. *This dialog requires the user to either choose an account to charge, or create a new one.*

He then receives two receipts, identical except that one provides a space for a signature from the customer. This prevents a customer from questioning his or her purchase of an item or items. We simply keep all the signed receipts until the balance has been paid. Every month we mail out a statement. At the end of the day the daily report lists a figure for House Accounts as a payment method. When the money in the drawer is counted, the figure listed under the "House Accounts Total Charged" heading can be subtracted from the Net Sale figure on the report, and the drawer should match that number.

Set In Stone

After dealing with the List Manager for a short period of time I discovered what I'm sure many veteran Macintosh programmers have long since known: The List Manager is somewhat of a joke. While it offers a host of useful features, it leaves a lot to be desired. The most obvious limitation is its inability to have differing widths among columns. Also, there can be only one style definition per list. In the case of my application, it was very difficult to have three columns in the queue and have more than a few characters of the item's name visible. To see more of the item's name, I had to enlarge that column and shrink another one (such as the quantity column, which needed only two or three characters). The only way to do this easily was to use StoneTable from StoneTablet Publishing. StoneTable also enabled me to use different styles for each cell. In particular, I wanted to align the price column flush right, yet I wanted the rest of the columns to align flush left. StoneTable proved to be a great enhancement to the user interface.

Employee Feedback

Without question, the most appreciated feature was the new on-screen queue of items. Even employees who previously had little experience with the cash register (production staff, for

example) found it easy to wait on customers if the store was so busy that we needed extra hands. The user interface proved to be incredibly intuitive and enabled a variety of functions to be performed on an item (return, discount, void, etc.) merely by selecting the item from the list and clicking the appropriate button. Additionally, employees mentioned how they appreciated that every time they clicked a button to add an item to the queue, it was automatically selected, thus they could perform a function on the most recent addition without having to make extra clicks. The greatest enhancement the list offered was scrolling, which was an inefficient and inhibiting task on the old system. Scrolling was now fast, and intuitive. Furthermore, there was now a method of determining if there were items above or below the "viewrect" because of the existence of the scroll bar.

Hardware

In truth, the most intimidating part of this endeavor was learning to communicate with point-of-sales hardware devices. I had no experience with hardware related issues, so there was an entire world of unconquered code awaiting me. I hadn't used the Device Manager or even OpenSerial() before. The three devices I needed to get my software to communicate with were receipt printers, cash drawers, and pedestal displays. I was very lucky that the manuals that came with each of these devices were thorough and certain sections were clearly written with programmers in mind.

In truth, communicating with the receipt printer wasn't as difficult as I had expected. The hard part was making the receipts look nice. The particular printer we chose was the Star TSP200, a thermal printer with an auto cutter. It's very quiet, very fast, and it can print bitmaps. It has a host of control characters that enable printing modes such as bold, inverse, underline, double width, double height etc. This enabled me to format the receipts to look very snazzy. Much thanks to Charles Rentmeesters (who wrote a Chooser level driver for the TSP200 and proved to be and endless source for information) I was able to print the logo for our store at the top of every receipt and report. Unfortunately, using the Chooser level driver was not possible because it did not provide a method for sending control characters to the printer. I was forced to send text directly to the printer via the serial port.

A few tidbits were added to the receipt that were not printed by the old setup such as the store's address and phone number, the payment method, cash tendered and change (if it's a cash transaction) and the account holder's profile (if it's a house charge). I spent even more time formatting the reports on the receipt printer because they make use of several columns, and with a 3" wide roll of paper, things can get a bit tight.

The cash drawer proved to be a cinch to use. The TSP200 has a pass through modular jack that connects to most cash drawers, among them the Loyal Cash Drawer we purchased. To open the drawer, you send a control character to the TSP200 and it in turn opens the drawer. It's that easy.

The device that was the most fun to work with was the pedestal display. These devices are little LCDs mounted atop a pole with a base. You've no doubt seen them integrated into cash registers at the supermarket. They provide the customer with information for each item entered into the register, and typically display the total sale amount when the items have all been entered. Our pedestal display was manufactured by ID Technologies. This unit connects to a serial port and can be operated in one of two modes: pass through or stand alone. When in pass through mode, the device displays data only when the data is prefixed with a user defined control character. The rest of the data is passed on to the next device in the chain (typically the receipt printer). Because the Macintosh has two serial ports this mode was not necessary. However, certain features, such as scrolling messages, were available only in pass through mode. Thus, while I had a serial port available, I used pass-through mode. (See **Listing 1** later in the article.)

As with the receipt printer, communicating with the pedestal display was essentially just a matter of streaming text through the serial port, but it was fun to use the control characters and to spice things up with scrolling messages among other things. The unit has two lines with 16 characters on each line. I designed the output to show the last item entered into the queue on the first line and the running total on the second line. The old system displayed only the price of the last item entered and wouldn't show the total until you opened the drawer. It also displayed only numbers.

Bigger Isn't Always Better

The icing on the cake was the monitor. We didn't really have room for a CRT, so I investigated LCD flat screens. I've always been charmed by the flat panel displays on laptops, but I didn't think of using one until I realized that we were going to have to make this system fit on a countertop (an afterthought for a software engineer). We ended up with CTX 12.4" LCD displays. After adapting them for touch sensitivity, we had a three inch deep screen with a very nice tilt base. Better yet, these screens could run at 800x600 (which gave us plenty of room for the on screen keyboard, etc.). The touch screen is perfect for the user interface; it ties everything together. The keyboard sits right on top of the cash drawer in front of the screen and receipt printer. Luckily, the dimensions worked out perfectly. (See **Figure 6.**)

Figure 6. *This is one of our cash registers. This computer is stored on a shelf below the counter.*

Some Assembly Required

I spent more time researching the hardware for this system than I did writing the software. This is primarily because most POS systems are integrated "all in one" systems. These systems have many advantages, however they are typically very costly. It is much cheaper to assemble a system by hand, but finding the right retailers can be a time consuming process. After spending many hours in Yahoo and other search engines, I did find a number of POS retailers who were willing to sell individual components such as receipt printers, pedestal displays, cash drawers, barcode scanners, magnetic strip readers and supplies. Finding the right retailer is the key. In particular, I was looking for one that was familiar with and, more importantly, focused on integrating POS hardware with the Macintosh.

Finding an LCD monitor manufacturer also proved to be an arduous task. I communicated with at least three or four companies. I actually drove to visit one manufacturer and almost purchased their screens. That was before I found another manufacturer who provided me with almost identical screens for half the price. Watch out, it's easy to get burned when building a system piece by piece.

For those readers interested in putting together a similar setup, here are some prices you can use as a frame of reference. A cash drawer shouldn't be more than $200 - $300. Receipt printers can vary in price based upon their printing mechanisms. Dot matrix printers are obviously much cheaper than thermal printers. A dot matrix machine should cost $400 - $500. Thermal printers are typically about $700 and can jump up into the $1000 range. Pole displays vary in price depending upon the number of characters and rows they display. In general, prices are between $300 and $400. As is typically the case, purchasing in bulk will almost always yield lower prices.

The glue that holds the system together obviously is the computer itself. Now that clones are available there are some choices, but the field is fairly limited and prices are almost identical among mail order companies. We bought the cheapest machine available. Speed certainly isn't an issue for a POS system, and no fancy features are necessary. The slowest and most obsolete machine available could still run a POS system with no noticeable loss of performance.

Let's Get Technical

All right, enough with all the talk. It's time to blow away some of the smoke and reveal how to make a Macintosh work with all of this hardware. **Listing 1** demonstrates how to send data to a serial device such as the pedestal display. Communicating with a receipt printer is an almost identical process, but obviously it requires formatting the page with tabs and other control characters ('\t' doesn't work). There is nothing earth shattering here for programmers experienced with the Device Manager or Serial Drivers. Mere mortals like me, however, have little experience with device communication outside of the Chooser; thus this may shed some light on the basics.

Listing 1: PDisplay.c

PD_Item

This routine displays the last item added to the queue on the first line of the pedestal display. On the second line it displays the current total for the pending sale. This routine is designed to work with the display unit when it is in "pass through" mode. In "stand alone" mode, simply precede the desired output string with "\r\r" to clear both lines of the display, and then send both lines one and two separated by a '\r' as one string rather than sending one line at a time with "!#1" and "!#2" preceding each string. This routine takes a character pointer item_name, and a double current_total as parameters. It returns TRUE if it successfully displays the item and total, and FALSE if it is unable to communicate with the device.

```
Boolean PD_Item(char *item_name, double current_total)
{
  short  err;
  short  aout;
  long count;
  char buffer[20];  //leave room for a 16 characters plus a
                    //few extra control characters

  err = OpenDriver("\p.aout",&aout);  //open the modem port
                                      //for data output

  if(err)
    return FALSE;
  //pedestal displays usually use 2 stop bits as opposed to 1
  //and can only communicate at 9600 baud
  err = SerReset(aout,baud9600+stop20+noParity+data8);
  if(err)
    return FALSE;
  sprintf(buffer, "!#1%.16s", item_name);
  //"!#1" tells the unit to clear the first line of the
  //display and then set it to the following 16    //characters.
  count = strlen(buffer);

  err = FSWrite(aout,&count,buffer);
  //send the data out to the port, for async communication,
  //PBWrite is better, but this isn't a time consuming
  //transfer of data, so FSWrite is adequate

  if(err)
    return FALSE;
  sprintf(buffer, "!#2Total - $%.2f", current_total);
  //as before "!#2" tells the display to set the second line   //of the display to the next
16 characters that follow
  count = strlen(buffer);
  err = FSWrite(aout,&count,buffer);  //send the data

  if(err)
    return FALSE;
  CloseDriver(aout);   //close the modem port

  return TRUE;
}
```

Conclusions

I don't see my software as a particularly marketable product, it's far too customized and doesn't facilitate the needs of a large business. It was designed for a small business that sells goods they either produce or package themselves. I suppose that's why most of the other software I looked at didn't seem right for us. I looked around for a piece of software that could accommodate our needs, but all I found were applications designed to be used with barcode scanners and PLU numbers. Clearly we have a different type of operation. Of course, few "mom and pop" stores have an in-house programmer who works for free. The point is that if there's anything I learned from this, it's that the old adage

is true: use the right tool for the right job, even if it means building the tool yourself. Oh yeah, I did learn one more thing: no good piece of software beeps on a regular basis.

Online Resources

For those readers interested in assembling a similar setup, here are several web links which should offer helpful information. Please feel free to contact me for further information or more online resources if you need them.

Retailers

- POS Direct, <http://www.posdirect.com/>
  Not the greatest web site I've seen, but they have excellent customer service and a good downloadable catalog. Very helpful with Macintosh implementation.
- Direct POS, <http://www.directpos.com/>
  A bigger selection of equipment, but less tailored to use on the Macintosh.

General Info

- POS International, <http://www.posintl.com/posdex/informed.html>
  This site offers a glossary of terms, and provides some helpful information on many different POS peripherals.

Receipt Printers

- Star Micronics, <http://www.starmicronics.com/product/receipt/index.htm>
  Star is without question the most popular manufacturer of receipt printers, and they offer a very nice line of printers from entry level to high quality thermal printers. Macintosh Chooser level drivers are available from POS Direct for a steal.

Touchscreens — LCD & CRT

- Pixel Touch, <http://www.pixeltouch.com>
  Good selection of both LCD & CRT screens with good customer service. Reasonably priced.
- PixelVision, <http://www.pixelvision.com>
  Cutting edge LCD innovation, but at a price. The only available 16" LCD screen on the market is manufactured by these folks (and boy is it nice).

Pedestal Displays

- ID Technologies, <http://www.idt-net.com/1800.htm>
  ID Technologies has a very reasonably priced pedestal display, as well as other products such as magnetic strip and barcode scanners. MT

Visit MacTech® Magazine's Web site!
http://www.mactech.com

Credit Card
Authorization
and
Electronic Deposit Software
800-4-TELLAN
www.tellan.com
Tellan Software, Inc.

The Quest for a web front end to Oracle on a Mac...

ends with X_20.

Details and free demo at www.xperts.com
Custom development available.
sales@xperts.com
800.356.8040


XPERTS

by Jeff Ganyard

Electronically Distributing your "Killer App"

A quick look at electronic software distribution solutions for the Mac OS

So you're sure that you've finally built the "killer app" all you need is a way for the rest of the world to realize it and buy it. Now you've got to get that marketing machine fired up, educate your sales force, start advertising, book booths at trade shows and most importantly, distribute that all important *Internet demo* version.

All right, but how are you going to get that demo version to compel the users out there to buy your software? How can you offer enough of the full application's features so that everyone wants it, but not so much that you cannibalize sales of the commercial product? The balance between features and limitations on "demoware" can't be found by a simple formula but you may want to look into Electronic Software Distribution (ESD) solutions. These products can help you easily build limited function, "trialware", "nag ware" or time restricted versions of your software. And when the customer makes the decision to buy the full product, they make their purchase and unlock the software that they already have on their hard disk.

Here's a quick overview of four of the ESD products for Mac OS: InstallerMaker from Aladdin Systems, InterLok from Pace Anti-Piracy, ZipLock from Portland Software, SalesAgent from Release Software Corporation, and Kagi, an electronic software payment service. The first thing you must do is identify the ESD model that will work best for your products — each company uses a somewhat different model.

InstallerMaker

With the release of InstallerMaker 4.0, Aladdin Systems has taken their successful installer creation technology and added transaction capabilities to its feature list. There is no up front fee to create trialware with InstallerMaker. Their model is based on collecting fees from each product sale, starting at a minimum $5 per transaction and moving to a percentage in the range of 10-15%. (The exact amount is somewhat negotiable; contact their sales organization for details.) InstallerMaker is very easy to use, yet powerful enough to meet most people's needs.

Not surprisingly, the user interface looks and acts much like an extended version of StuffIt Deluxe, but with much more control over what can happen while the archive is being decompressed, files placed in required locations, separate items requiring registration keys, etc. InstallerMaker provides control over the number launches, the number of days until expiration or a fixed expiration date. The user is greeted with a dialog at every launch with information on how much longer the demo will be useful and options to purchase, register, or to use as a demo. Aladdin accepts payment transactions by phone, postal mail, fax and over the Internet using their own protocol. Aladdin uses it's own encryption technology for both the software and the Internet transaction.

Aladdin will provide daily reports on activity and monthly accounting. And of course, since it is InstallerMaker, there is the added bonus of a full featured installer combined with this ESD solution.

InterLok and InterLok Pro

Pace Anti-Piracy offers a very different approach. They have created what should be more accurately described as a developer

Jeff Ganyard is the co-founder of Mac ISP, supporting people and organizations using the Mac OS to build, develop and provide Internet services. He is also a contributing editor for this issue of MacTech Magazine. Jeff was formerly an Internet Evangelist for Apple Computer, Inc.

ool for distribution rather than an transaction system. They offer their product in 2-tiers; InterLok and InterLok Pro. They base their model on quantities distributed per year and are not involved in processing or taking a percentage of the actual financial transaction. The process begins with their wrapper tool, allowing you to easily create distributable products with control over the number of launches, expiry after a set number of days or a fixed expiration date.

When the customer decides to purchase the software, a challenge-response mechanism is used. The wrapper technology will build a challenge based on a variety of elements. The challenge (and presumably the payment) is then sent by the end user to you, where you provide the response to unlock or register the product. Because this mechanism is variably generated by environment, the need for encryption is dramatically reduced, except, of course, for the financial transaction. Their goal is to provide ESD and serialization support for your product, while you take care of distribution and financial interactions. Their key technology can be associated to a hard drive, a key diskette or a simple pass phrase. They look at the characteristics of the media when the software is installed. This means that their unlock authorization and serialization will continue to be valid even after the hard drive (or other media) has been initialized.

Since you completely control the Satisfaction tool – the tool that generates the response codes – the transaction method is up to you. Phone, fax, email are obvious choices, and because Satisfaction is fully scriptable, the construction of a custom CGI to process orders is fairly simple. Release has included a set of docs on scripting the product along with sample web pages and a sample AppleScript CGI to get you started.

The Pro version costs almost twice as much as the standard version, but adds more robust encryption, support for non-executable files, network copy protection, key diskettes, limitation of application functions, installer integration and more flexibility of controls than the standard version though its own API. With the Pro version you could easily build a demo, trial, basic and a "pro" version of your product all from the same source. As a matter of fact, InterLok and InterLok Pro are an example of this. This would allow your customers to upgrade to a more full featured version of your product just by entering a new response code (that they have purchased from you, of course). Look for a Windows version coming around the end of this year.

ZipLock

Portland Software recently released version 2.0 of their product ZipLock. This version adds support for Mac OS and Newton products. They provide an end-to-end solution, where you can use their service for your transactions or you can buy their server software to handle your own merchant account transactions. Their approach is to build an ESD channel and provide the tools for you to participate. The first step is enter into a relationship with a clearing house, their current clearing house partners are: CyberSource Corporation, Internex PowerCommerce Clearinghouse and LitleNet, LLC. With that established, you will be able to use their builder application. (As of August 1997, they were working on a test server for demo use.) The distribution model is buy before you try, but they include the function for a 30 day money back guarantee to the end user. Once the builder has been used on your application, it can be easily distributed online or offline, or through whatever other means you wish. The clearing house will then charge a fee for each transaction, typically a dollar or two.

Portland Software uses RSA licensed encryption in both their package encoding and the purchaser to clearing house transmission. There is no need for a secure connection because the contents are already encrypted. They have gone to great efforts to hide the key information from the user. Once the user has downloaded the product, and tried to launch it, it will first verify the integrity of the package; when successful, the user is greeted with a set of marketing or questionnaire screens. One of the strengths of Portland's product is the complete customizability of these screens. When the user is ready to purchase, they must be connected to the Internet, the client product uses http to communicate with the server to perform the transaction.

If you are interested in purchasing their server software and becoming your own clearing house, they have that option. It is pricey, coming in at $25,000 and is available for Windows NT and Solaris only. You will probably want to expect some very significant sales figures to consider this option.

MkLinux

Microkernel Linux for the Power Macintosh

Why wait for Rhapsody? MkLinux is a complete system, based on Linux 2 and the Mach 3 microkernel. It includes a complete software development system (gcc, gdb, perl, ...), X11R6.3, and hundreds of other commands. Get MkLinux and start working right now with Mach, Objective-C, UNIX development tools, and more...

MkLinux builds and runs most Intel-based Linux software. It works efficiently and reliably on a wide range of Power Macintosh platforms, with other ports on the way. MkLinux has an active user community and a substantial Reference Release, both sponsored by Apple Computer. If you want to get a jump start on Rhapsody, this is a great way to do it!

Visit Apple's MkLinux web site (www.mklinux.apple.com) and Prime Time Freeware's web site (www.ptf.com) to find out more about MkLinux and other fine freeware products.

Prime Time Freeware
370 Altair Way, #150
Sunnyvale, CA 94086

info@ptf.com
(408) 433-9662
(408) 433-0727 fax

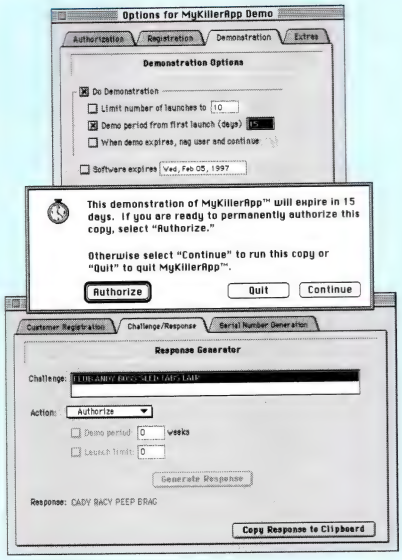
InterLok Pro™
by PACE Anti-Piracy

Trialware. Demoware. Internet distribution. You know you need it. You know it has to be flexible and secure. And you don't want to write any more code, or give a percentage of your sales to one of those "service bureau" solutions. Sound familiar? Then take a look at the InterLok Software Authorization system.

InterLok provides "software registration insurance", letting you turn finished applications into locked, time-limited "trialware" in minutes -with no additional programming! Your software can then be distributed via any media (Internet, CD-ROM, diskette), used under demonstration limits that you define, and remotely unlocked with a unique key that you issue.

For the most flexible, secure, and powerful software authorization system, turn to PACE Anti-Piracy.

Secure Software Distribution on the Web



Features

- No source code modifications required
- Automatically locks data files to run-time database solutions
- Works with all leading installer tools
- Demo options include days of use, specific date, number of launches, unlimited demo, and expiring to "nagware"
- Authorization options include serialization, machine-unique password, and uncopyable key diskettes
- Customizable text messages for user dialogs
- Adds registration information capture to your application with no coding
- Available API for use with "non-application" components, such as system extensions or plug-ins.

Lock in your software revenues with InterLok!

Price listed is for InterLok Standard/3000 edition annual license. See PACE web site for other prices.

InterLok Standard: Price $499.00

PACE ANTI-PIRACY

Visit our Web site at:
www.paceap.com
Vox: 408-297-7444
Fax: 408-297-7441
E-Mail: sales@paceap.com

SalesAgent

Release Software provides a turnkey solution. You send them your product, they wrap it with their SalesAgent technology and will either send it back to you for you to distribute or they will distribute it online, or both. This allows online and offline distribution for you. Currently their Mac OS product produces only "buy before you try" packages, but they intend to offer trialware capabilities in a release due at the end of this year. All of the sales information is available in real time through a secure web site through their Sales Manager program.

In addition to phone and fax, they accept transactions via HTTP or a direct modem dial-up — convenient for customers who do not have a connection to the Internet. There is an initial "wrapping" fee of $500 per product and they will then take 15% of each transaction. Their technology places a SalesAgent on the end user's hard drive, to prevent unauthorized redistribution of the software. The same SalesAgent will handle all products wrapped by Realease Software. Since the transaction is controlled on their server, price changes can be made dynamically, however the price indicated in the distributed package cannot be changed. Their encryption is licensed from RSA and they use it for their wrapper and the online transactions. They also have received permission to export this use of RSA encryption. Because the user has to pay before accessing the package, it also well suited for non-executable and data file use.

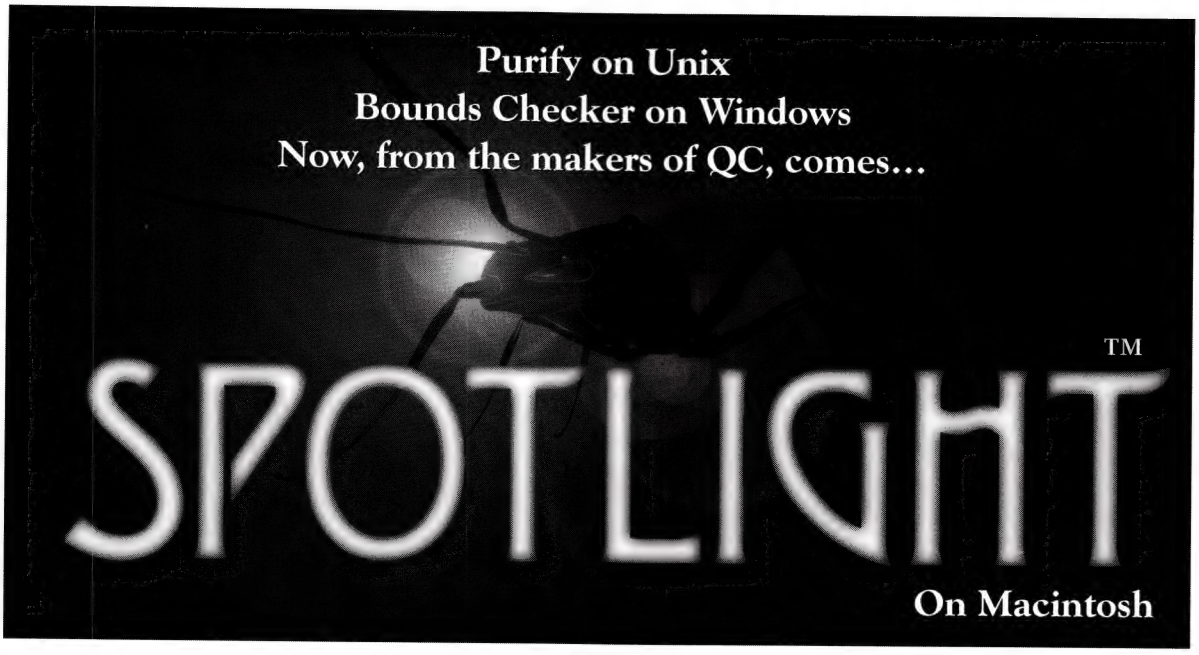
Kagi

Another alternative is a payment processing service such as Kagi. Kagi is a popular service used by many shareware authors. They will help you distribute the software and collect payments, but they leave the serialization implementation up to you. Kagi takes 6.5% of the sale price and passes on all applicable payment service fees; for example, Visa transactions cost an additional 3.5% and a $.032 transaction fee.

Conclusions

There are a variety of products and services available to help you distribute your software electronically. Each has its own strengths and weaknesses. To choose the best for your needs, you must carefully research which distribution and transaction models suit your product and your business model. You can find more information to help you with your research at these web addresses:

1) Aladdin Systems, Inc. at <http://www.aladdinsys.com>.
2) PACE Anti-Piracy at <http://www.paceap.com>.
3) Portland Software, Inc. at <http://www.portsoft.com>.
4) Release Software Coporation at <http://www.releasesoft.com>.
5) Kagi at <http://www/kagi.com>. MT



Find Bugs Fast

Spotlight is the first **Automatic Memory Debugger** for the Macintosh. Instantly detect invalid memory accesses, bad toolbox parameters, leaks, stack overwrites, memory relocation problems, and much more.

Spotlight uses your XSYM file to automatically patch your application — no need to change your source code. No need to recompile. **No learning curve whatsoever.**

The interface gives you instant feedback when an error is detected. You can ignore the error, ignore all future occurrences of the error, or log the error to a text file for later analysis.

Fine Grained Memory Protection

Spotlight identifies reads and writes outside of the application and system heap. But it does not stop there. Spotlight's Object Code Replacement instrumentation **inspects each read and write instruction in your code,** detecting faulty memory reads and writes that occur between blocks, in released blocks, across multiple blocks, and in ROM.

This technology works on any heap object you can allocate: Mac heap objects, C 'malloc' heap objects, even C++ objects created with new.

Spotlight stops your application whenever a faulty memory access is about to occur and displays the **exact source code line** where it will happen. The calling stack is shown along with a display of all variables. A memory viewer shows the erroneous memory address and contents.

Toolbox Validation

Spotlight checks parameters to **over 400 toolbox API calls,** automatically validating handles, memory blocks, return values, and so on. Specific checks catch subtle errors such as drawing into an unlocked GWorld, passing an invalid window pointer, passing an address within an unlocked handle to a routine that may move memory, and too many more to list.

Leak Detection

No more struggling with MacsBug. On program exit Spotlight provides a **clear listing of all leaked memory** showing a full stack trace from where the memory was allocated. Instantly discover all leaked Mac OS objects, malloc'd objects, C++ objects, and resources.

Limitations

Spotlight requires an XSYM file to function. MPW and CodeWarrior users can generate these directly. Symantec users must use ToolServer to link with an XSYM capable linker. Spotlight requires a PowerPC processor.

Spotlight works on the object code. **No source code is required**: just the thing for testing purchased third party libraries.

Availability and Pricing

Pricing for Spotlight DR1 is $199 US (plus $5 shipping and handling within the continental US, $15 for international orders). This includes a free upgrade to the GM version and access to ftp interim upgrades leading up to GM. QC users can cross-grade to Spotlight for only $149.

All Onyx products carry a 30 day no questions asked money back guarantee.

Onyx Technology, Inc.
7811 27th Avenue West
Bradenton, Florida 34209

sales@onyx-tech.com www.onyx-tech.com
941 795-7801 941 795-5901 (fax)

Spotlight and QC are trademarks of Onyx Technology, Inc. All other trademarks are property of their respective owners.

by Dr. Scott B. Steinman

High-Speed Inter-Computer Coordination

Combining C++, FrontierScript, LabVIEW and multiple Macs to solve a real world problem

Introduction

A few years ago, I was asked to put together an infant vision clinic that would offer not only state of the art tests for assessing the function and health of an infant's visual system, but which would also be fast, easily operated and flexible enough to readily allow future expansion. It didn't take much thinking to come to the conclusion that only the Macintosh could do it. But the programming was far from straightforward at the time, since some of the system software that I needed had only just become available and was not well documented.

This article will discuss the specific programming issues that had to be solved in order to achieve the design goals of the clinic, as well as many of the workarounds that had to be made. These issues also apply to other medical, scientific and engineering tasks. We'll see that sometimes a seemingly more roundabout design actually produces a simpler, faster and more flexible solution.

Design Decisions

The infant vision tests typically involved displaying a rapidly animated visual stimulus that moved targets on a screen or changed the target's contrast dynamically. *At the same time*, physiological data such as eye movement recordings or evoked potential recordings ("brainwaves") have to be obtained in response to the stimulus. Both tasks need to be executed rapidly — the animation may be run at refresh rates up to 60 Hz, while the data acquisition may occur at up to 1000 Hz.

While speed and flexibility are always two important design goals, the system had additional restrictions that made these goals even more critical. First, the time required to conduct the full test sequence needed to be *extremely short* since our subjects were very young infants whose attention span is limited in duration. The examiner has to collect data extremely rapidly on any given test, then quickly proceed to the next test. Second, the same examiner who has to control the data acquisition also has the responsibility of directing the infant's attention to a visual stimulus. This not only reinforced the need for speed of testing, but also *ease of operation*. Finally, the software system needed to be modular and *easily extended* when new tests of infant vision were introduced.

I chose National Instruments' LabVIEW™ as the language in which to write the data recording and analysis code. LabVIEW is a true "visual" (that is, iconic) dataflow programming paradigm, much like Prograph CPX™. Visual programming allows rapid application development (RAD) in a fraction of the time required to write an equivalent C/C++ program (for example, see Steinman & Carver, 1995). LabVIEW also includes a huge library of data acquisition and data analysis code modules (called Virtual Instruments, or VIs for short) that may be used to build a

Dr. Scott Steinman is a vision scientist and Chair of the Department of Biomedical Sciences at the Southern College of Optometry. He develops software for clinical vision testing, research and education. Scott programs in C++, LabVIEW, Frontier, Prograph and Java. He has published numerous articles on computer programming, two of them previously in *MacTech*. You may have noticed his book, "Visual Programming with Prograph CPX", on sale in the *Developer Depot* (do I have to hint more than this?). You can reach him at steinman@sco.edu.

recording system. In addition, LabVIEW's integrated user interface design tools provide a uniform intuitive "front panel" of controls that can be easily operated even by assistants.

Unfortunately, while LabVIEW is well suited for laboratory data acquisition and analysis, it is not capable of performing acquisition and analysis while *simultaneously* generating rapid animated visual stimulus displays. One solution that was considered was to call Macintosh Toolbox and QuickDraw routines via external C language external code modules, but these graphics routines could not be called *in parallel* with the data acquisition VIs. Even if these routines were called in separate programs, running all of these tasks on a single machine could still be risky since processor interrupts of the data acquisition hardware and those of the visual stimulus animation code could potentially interfere with each other — that is, data samples could be "skipped" during graphics operations or graphics could "shear" when data samples are acquired.

We therefore chose to implement the system on *two* Macintosh Quadra computers — one solely for recording and the other solely for stimulus display. This would allow each computer to perform its single specialized task at maximum speed without interfering with each other.

Both the recording program on one computer and the stimulus program on the other computer had to "know" what the other was doing at any given moment. Their operations needed to be tightly *synchronized* if the two-computer system were to act as a single unit. The overall operation of the laboratory software would be controlled by the examiner on the recording computer, and the second computer would act as a "slave" to the recording computer. This computer would act only in response to commands sent to it by the "master" recording computer.

This article will discuss the specific programming issues that had to be solved in order to achieve real-time inter-computer coordination while meeting the design goals of the clinic. I'll discuss many of the workarounds that needed to be made. These issues also apply to other medical, scientific and engineering tasks. We'll see that sometimes a seemingly more roundabout design actually produces a simpler, faster and more flexible solution.

Clearly, two forms of communication were required. Each would be executed independently. The first was to maintain tight control over the timing of the sequence of events during data acquisition. Digital I/O lines (NB-DIO-96, National Instruments) joined by a ribbon cable were used to transmit these signals.

The second form of communication was to transmit commands from the recording computer to the stimulus computer to "tell" it what stimulus to display and when. This would, of course, involve Apple events. These Apple events would be sent across an EtherTalk network cable that interconnected these two computers alone. In initial testing of the software system, it was found that LocalTalk was too slow. When Apple event commands were sent to the stimulus computer, the recording computer often had to wait for a reply that indicated that the command was received correctly. An Apple event could "time out" while waiting for this reply if the transmission time was too long, locking up the data acquisition when the recording computer kept waiting for the reply. The much quicker EtherTalk was the solution to this problem, as replies could be received fairly instantaneously.

Finally, two more decisions related to programming language had to be made. The first was which programming language to use for the stimulus display software. LabVIEW would not do. While LabVIEW has some capabilities for sending Apple events (with provisos to be mentioned below), it has extremely limited abilities to respond to Apple events. C++, on the other hand, is well suited for writing Apple event handlers, but also allows for rapid interrupt code for accurately-timed stimulus display animation (see Steinman and Nawrot, or Steinman for such graphics techniques).

Although Apple events would be involved in the inter-program communication between the data acquisition and stimulus display computers, they could not be used in a uniform manner throughout the software due to limitations in LabVIEW's inter-program communication code. One possible solution to this problem would be to use AppleScript, but for several reasons to be discussed below, inter-program communication was implemented with the UserLand Frontier™ scripting language (now available free at www.scripting.com/frontier).

It could be argued that the ethernet connection could be replaced by a simple high baud rate serial line that carried messages from one computer to the other. However, there is one disadvantage to doing so. In the current system, both the LabVIEW recording program and the C++ stimulus display program are not aware of whether they reside on two computers or on a single computer. One side effect of using Frontier as an intermediary for passing commands is that Frontier hides these details from the LabVIEW and C++ programs. This means that if the present stimulus generation and recording system could be implemented on a single computer in the future, all that would need to be done is to edit one subset of Frontier scripts — neither the recording software nor stimulus display software would need rewriting. If a serial line had been used instead, all of the inter-program communication code would need to be completely rewritten.

Software System Components

For a better understanding of the workings of the software, the general infant testing sequence is as follows: The main program establishes the connection between the recording and stimulus computers across the EtherTalk network. The digital I/O lines are then initialized. At this point, the examiner selects a test to be run.

A command is then transmitted to the stimulus computer to launch a particular stimulus display program, whose stimulus parameters are transmitted to the display program by a second command. The examiner then clicks a button on the recording computer screen to start recording. A digital signal is sent to the stimulus computer to enable the stimulus animation, and the stimulus computer returns a digital signal to trigger data acquisition, time-locked to the stimulus display. After data collection is completed, commands are sent to the stimulus

computer to stop the stimulus presentation, then quit the stimulus display program.

The software system has been designed to be both flexible and modular. It is composed of three major components: The first, implemented in LabVIEW™ on the recording computer, initiates an Apple event link between the two computers, initializes the data acquisition and digital I/O boards, allows the examiner to select which to run and specification of stimulus and recording parameters, as well as starting, pausing or halting data acquisition. A uniform user interface across all tests simplifies operation of the tests.

Visual stimuli are presented via the second component, a collection of very small programs written in Metrowerks CodeWarrior C++ and Mathemaesthetics Resorcerer™. These programs take advantage of a reusable code library of drawing and animation routines that allows both palette animation (see Baro) and frame animation (see Steinman and Nawrot, or Steinman). These programs are small because they perform only three chores: (a) receive Apple event commands, (b) display animated graphics, and (c) read and write to the digital I/O lines for synchronization with the recording software.

The third component coordinates the stimulus generation and data acquisition via Apple Open Scripting Architecture-compatible scripts in the Frontier™ scripting language.

Inter-Program Communication Problems and Solutions

With a combination of different programming tools, several problems that are specific to real-time simultaneous visual stimulation and data acquisition have been solved. Specifically, we will present an easy way to synchronize two Macintosh computers to work as a single laboratory device, via software commands and hardware signals passed between.

The first problem which we had to confront and solve dealt with shortcomings in LabVIEW's capacities for inter-program communication. While Apple event support is included in LabVIEW, it is mostly specialized as VIs that are used to execute other VIs, such as *AESend Run VI, AESend Open, Run, Close VI, AESend Close VI, AESend Abort VI* and *AESend VI Active?*, or responses to these commands. For programs composed of code other than its own VIs, LabVIEW is more capable of responding to commands than sending them. A few other Apple event-related VIs exist, which are geared towards starting or quitting other programs, but these are not sufficient for laboratory program control.

Fortunately, a LabVIEW VI that is often overlooked by programmers just happens to be the one VI that is critical for controlling the operation of the stimulus graphics programs by the recording computer. This VI is called *AESend Do Script*. As its name implies, it is specialized towards sending scripts. When such scripts are received by a target program, their text must be decoded into a series of instructions to be carried out by that program.

The next decision was to choose a specific scripting language. AppleScript presented several obstacles. The first is that while AppleScript is relatively slow, even on PowerPC-based Macintoshes. The second problem is that AppleScript was not primarily designed for sending commands quickly over a network to a second computer. To send an AppleScript or Apple event across a network, the PPC Browser must be invoked. The PPC Browser is intended to allow users to choose which computer and application should be sent a command, but it also has one shortcoming for our purposes. It provides a level of security via the User Identity dialog box. While this is useful for preventing unwanted connections across the Internet, it is a severe impediment to our design goals. Every time we need to send an AppleScript or individual Apple event to each stimulus display program, we will be faced with the PPC Browser and User Identity dialog boxes! This is not only disruptive to the examiner using the software, but will also slow down our data collection.

A third problem is that if only AppleScripts are transmitted to the stimulus computer to initiate a stimulus display, these stimulus programs must be capable of receiving and parsing the AppleScripts. This presents a very difficult task for several reasons: (1) AppleScript programming is not entirely intuitive. (2) AppleScript programming requires programming knowledge about 'aete' resources that identify the Apple events "understood" by the receiving program. (3) In the present software system, this would require adding AppleScript support to each of a dozen small stimulus display programs. While adding AppleScript is a fruitful option for single large commercial programs, the time and effort required to add AppleScript support to several small specialized laboratory programs is simply not cost effective. We have selected an alternative that is simpler, yet overcomes many of the limitations of AppleScript.

That alternative is *UserLand Frontier™*. Frontier can transmit sequences of Apple events across a network at least ten times more rapidly than AppleScript scripts, and does not require the creation of 'aete' resources.

In our system, Frontier is installed on both the recording computer and the stimulus computer. When a command must be sent from the LabVIEW program on the recording computer to a C++ visual display program on the stimulus computer, it is first sent as a script from LabVIEW to a "master" copy of Frontier on the recording computer. This Frontier application is "told" to run a script stored within its Object Database that essentially transmits the command contained in the LabVIEW script across the Ethernet network to a "slave" copy of Frontier on the second computer. This second copy of Frontier translates the command into Apple event format and relays the command to the stimulus program, where the command is carried out.

Why use such a circuitous route? Why not just send scripts directly from the LabVIEW recording program to the stimulus display program on the second computer? Two reasons have already been mentioned: (1) Frontier speeds up the transmission of commands across the network, and (2) to avoid the need to add AppleScript-parsing code to each stimulus program. Let us add a to more important reasons to use Frontier as an intermediary in passing along commands: (3) Frontier simplifies

our programming task by translating the original command from LabVIEW's *Do Script* VI, which is in textual script format, into a form that can be handled with simple code in the stimulus programs — Apple event handlers. (4) Because Frontier itself sends the commands across the ethernet network, the intrusive PPC Browser and User Identity dialog boxes are avoided during the time-critical portions of data collection.

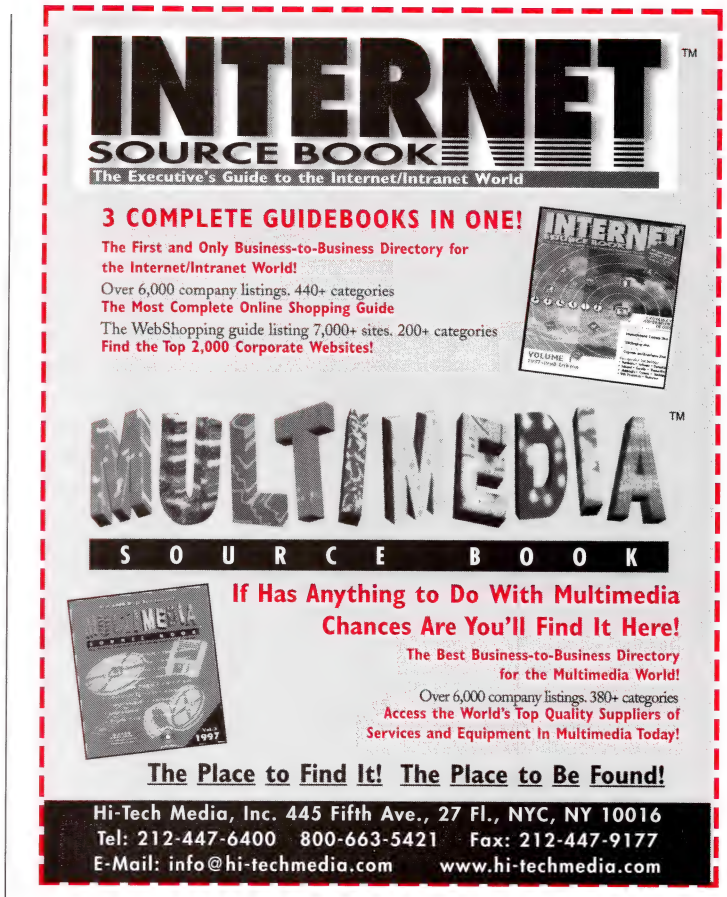
When the LabVIEW recording program must find a program on the stimulus computer to which to "connect" and send commands, it connects to the "master" copy of Frontier on its own machine. Similarly, the "slave" copy of Frontier on the stimulus computer connects to stimulus programs that reside on its own computer.. These two communication paths do not invoke the PPC Browser or User Identity dialog boxes, since each connection between programs is made within a single machine. The connection between the two machines is made by Frontier before the LabVIEW program is executed. Before launching the LabVIEW recording program, a small Frontier script is run on the recording Macintosh that results in the "master" copy of Frontier on that Macintosh establishing a connection across the network to the "slave" copy of Frontier on the stimulus Macintosh and telling that computer to do something trivial such as sounding a beep. This is the only time at which the PPC Browser and User Identity dialog boxes appear. During all subsequent experimental testing, these dialog boxes never reappear even when the experimenter switches between test types and stimulus types, because the connection across the network has already been established from one copy of Frontier to the other.

Let's examine the Frontier code that sets up the inter-computer communication across the network. **Listing 1** is a Frontier script entitled *testConnect* that first establishes the network connection between the two Macintoshes prior to the launching of the LabVIEW recording program. Each copy of Frontier contains an Object Database of frequently-used scripts and data that may be thought of as the commands that Frontier itself can execute, send or receive, as well as parameters for those commands. The *testConnect* script is stored in the Object Database of Frontier on the recording computer. The *testConnect* script simply sends a command to the stimulus computer to sound a beep. Because this is the *first* script to be transmitted across the EtherTalk network, it forces the PPC Browser and User Identity dialog boxes to be displayed at this time. This is beneficial because data acquisition hasn't initiated yet, so these dialogs cannot slow us the test sequence.

Listing 1: *testConnect* script

testConnect

```
« Establishes interconnection with copy of Frontier on second
« computer.This Frontier script on the "master" recording
« computer transmits a trivial script to sound a beep to
« Frontier on the "slave" stimulus computer to force the
« display of the PPC Browser and User Identity dialog boxes
« prior to test parameter selection and data acquisition.
« This script is simpler than that of Listing 3 because no
« parameters are transmitted with the "stimulus.beep"
« command.
« © 1997 by Scott B. Steinman. All rights reserved.

on testConnect()
   « Create local variable named "netScriptStr" to construct
   « text of new script

   local (netScriptStr)

   « Script will instruct stimulus computer to run script
   « named "speaker.beep" in Object Database of copy of
   « Frontier on stimulus computer

   netScriptStr = "speaker.beep()"

   « Construct script command:
   « 1. Create new script at location "scratchpad.netScript"
   « in Object Database of Frontier on recording computer.
   « This script will be to be sent to stimulus computer
   « 2.Tell Frontier that next few operations will work on
   « the newly-created script by setting target of Frontier's
   « operations to that location
   « 3. Clear contents at location "scratchpad.netScript"
   « 4. Set contents of location "scratchpad.netScript" to
   « text contained in netStringStr
   « 5. Reset Frontier operation target to the testConnect
   « script

   new (scriptType,@scratchpad.netScript)
   target.set (@scratchpad.netScript)
   op.wipe()  « Clear current contents of netScript
   op.setLineText (netScriptStr)
   target.clear ()

   « Send script command stored at "scratchpad.netScript"
   « across network to "slave" stimulus computer

   NetFrontier.runScript (@scratchpad.netScript,true)

   « Clean up

   delete (@scratchpad.netScript)
```

The testConnect script starts by creating a variable named netScriptStr to contain the text of the command to be sent to the "slave" copy of Frontier. This string is simply the name of a Frontier script ("speaker.beep") within the stimulus computer's Frontier Object Database that we want executed. NetFrontier has been designed to transmit scripts stored at a specific locations in the Object Database rather than transmitting raw text strings, so we need to transfer this command string into a new script, then transmit that script. In order to do this, we create a temporary entry named "netScript" in the recording computer's Object Database's "scatchpad" area to contain this text (its Frontier data type is, appropriately enough, scriptType). We then must copy the text of the script ("speaker.beep") from the netScriptStr string into the newly created script. We are now ready to transmit the contents of this newly-created script text across the network with NetFrontier's runScript command.

How is this testConnect script called to perform these actions if the LabVIEW recording program isn't running yet? Along with its own internal scripts in the Object Database, Frontier can create stand-alone, double-clickable scripts. A stand-alone script called ConnectMacs is executed by the experimenter before launching the LabVIEW recording program. It calls the testConnect script.

Experienced Frontier programmers might notice that we have added a second parameter to the runScript NetFrontier script to allow the option of waiting or not waiting for a reply during transmission of the script to the stimulus computer. This is because by default the Apple event Frontier command that sends information from Frontier to other applications waits for a reply from the receiving application. When we send commands to show visual stimuli just as we are about to begin data acquisition, we cannot afford the luxury of waiting for an Apple event reply, as this would delay the onset of the data acquisition. It is therefore imperative to avoid a reply by using Frontier's finderEvent command instead of Apple event to send the information; finderEvent by default does not wait for a response. This command was originally intended for sending scripts from Frontier to the Macintosh System 7's scriptable Finder, but it also suites our purposes well. The modified script is shown in **Listing 2**.

Listing 2: *runScript* script

runScript

```
« Modified runScript Frontier script.This Frontier script,
« part of the NetFrontier suite, sends a script across a
« network to control a second Macintosh computer. It has been
« modified to allow the option of waiting for an Apple event
« reply or ignoring the reply.
« © 1997 by UserLand & Scott B. Steinman. All rights reserved.

on runScript (adr,waitReply)
  on callback (netAddress)
    local (data,val)
    pack (adr^,@data)
    if waitReply equals true
      « Wait for reply from Apple event
      « (by default, all Apple event transmissions produce
      « reply in Frontier)

      if not AppleEvent (netAddress,'netf','inst',1,
       "scratchpad.netScript",2,data)
        return (false)
      val = AppleEvent (netAddress,'netf','runs',1,
        "scratchpad.netScript")
      AppleEvent(netAddress,'netf','dele',1,
                    "scratchpad.netScript")
    else
      « finderEvent command sends Apple event but
      « ignores reply

      if not finderEvent (netAddress,'netf','inst',1,
       "scratchpad.netScript",2,data)
        return (false)
      val = finderEvent(netAddress,'netf','runs',1,
       "scratchpad.netScript")
      finderEvent(netAddress,'netf','dele',1,
       "scratchpad.netScript")
      return (val)

  NetFrontier.buddyLoop (@callback)
```

Once the inter-computer connection is made, we launch the LabVIEW recording program and set stimulus and recording parameters prior to data collection. Setting stimulus parameters requires sending commands and data across the EtherTalk network, as dictated by scripts sent from LabVIEW to Frontier within the recording computer. But LabVIEW's script transmission VIs expect to be sending commands by connecting to another program across a network, and we don't want that to occur. We want LabVIEW to communicate only with Frontier on the same computer, and let Frontier handle the communication across the network. We therefore have to force LabVIEW to establish a "connection" directly to Frontier within the recording computer. **Figure 1** displays LabVIEW's graphical code for locating this particular copy of the Frontier application, then sending a small test script to Frontier to confirm that the connect was made properly.

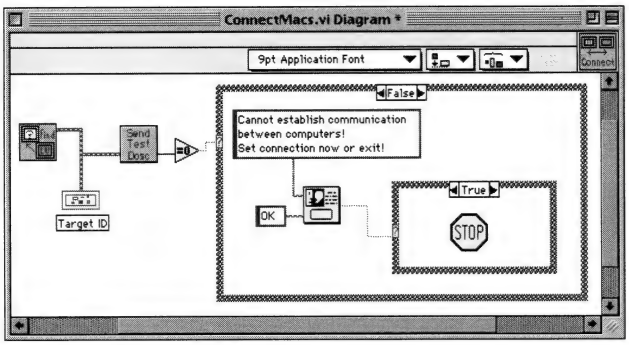


Figure 1. *LabVIEW ConnectMacs VI.*

The Find Frontier VI in **Figure 2** takes advantage of a LabVIEW PPC Toolkit VI named Get Target ID, which receives the name of the application to find and returns the its target ID, a LabVIEW structure (or "cluster" in LabVIEW terminology) that stores the location of a program on a network. We restrict the search to the copy of Frontier on the same machine hosting the LabVIEW recording program. The target ID returned by this search is used in all subsequent command transmissions to Frontier.

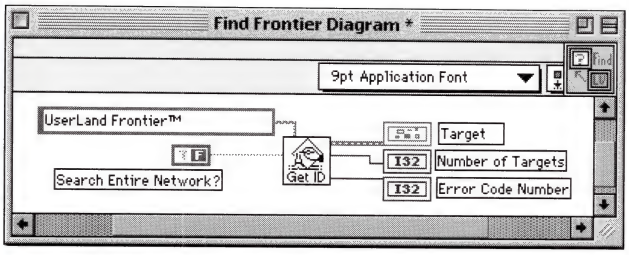


Figure 2. *LabVIEW Find Frontier VI.*

Now let's discuss the specific programming steps required to send commands and data from the LabVIEW recording program to their real target—the C++ stimulus display programs. In the example code to follow, the runStimulus command will be explained. This code is used once recording and stimulus parameters have been chosen by the examiner and we are ready to initiate data acquisition in response to a visual stimulus. The LabVIEW program initializes its data acquisition VIs, then sends a runStimulus command containing stimulus settings in the form of an Apple event to Frontier. Frontier then repackages the text contained within this Apple event into a Frontier script and transmits it. Upon receipt of the restructured script, the stimulus Macintosh's copy of Frontier executes the script, extracts the parameters of the script, and sends both a command and the parameters in an Apple event to the stimulus display program. An Apple event handler in that program retrieves the stimulus settings from the Apple event parameters, and displays the stimulus after sending a sync signal via the DIO lines to let the recording Macintosh know that it is time to start acquiring data. The net result is that the stimulus program only needs to receive an Apple event and its parameters in an easy-to-extract format, rather than a textual AppleScript that requires complex code to decipher.

The first step in this process is handled by the LabVIEW recording program. Once stimulus parameters have been selected and the experimenter is ready to record data, the Send RunStim Dosc AE VI is called (**Figure 3**) to tell Frontier to execute a script that receives stimulus parameters and then ships them to the stimulus computer. The script to be executed on the recording computer is named Stimuli.PVEP.netRunStim, and it receives as its argument the text contained in the script transmitted by LabVIEW. The script text contains the name of the Frontier script to execute — Stimuli.PVEP.netRunStim (a stimulus display program running a Pattern VEP visual stimulus) — and an argument to that script containing the list of the values to which the stimulus parameters are to be set ("50 380 8").

Visit MacTech® Magazine's Web site!
http://www.mactech.com

WEBsite Hosting Services
http://www.webcs.com

OPTIONS

Six accounts to choose from packed with features like CGI, secure server, audio support, FTP and TELNET logon access with plenty of storage space: all at stable and competitive rates.

Full internet merchant solutions and on-line payment systems make doing business on the web easier than ever before!

You've Found Your Web Home

WEBSTER COMPUTING SERVICES

SUPPORT

Full on-line docs, fast response customer support and available webpage design make your adjustment to our servers a breeze.

Your success on the web is our success. Personal and business sites stored the easy way: with NO HASSLES.

E-mail: pricing@webcs.com 24 Hour InfoFAX: 1-508-854-1789

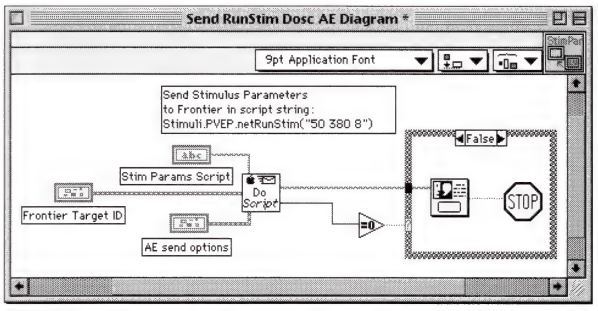


Figure 3. *LabVIEW RunStim Dosc AE VI.*

The netRunStim Frontier script is shown in **Listing 3**. The purpose of this script is to relay the command specified within the LabVIEW script to the stimulus computer, that is, a command to prepare a stimulus display, along with the desired stimulus parameters received as a single string argument to the netRunStim script named "str". This is done by packaging the command and stimulus parameters into a form that may be transmitted across the network by NetFrontier.

Listing 3: *netRunStim* script

netRunStim

```
« Frontier script within the recording computer's Object
« Database for interpreting the LabVIEW runStimulus script
« and forwarding it as a Frontier script to the stimulus
« computer.The command extracted from the LabVIEW script
« and the arguments for that command are transmitted
« separately by Frontier to the stimulus computer, using
« the NetFrontier.broadcast and NetFrontier.runScript
« commands, respectively. Returns true if successful,
« false otherwise
« © 1997 by Scott B. Steinman. All rights reserved.

on netRunScript(str)

   « Create local variable named "netScriptStr" to
   « construct text of new script

   local (netScriptStr)
   netScriptStr = "Stimuli.PVEP.runStim()"

   « Construct script command:
   « 1. Create new script at location "scratchpad.netScript"
   « in Object Database of Frontier on recording computer.
   « This script will be to be sent to stimulus computer
   « 2.Tell Frontier that next few commands will operate on
   « the newly-created script by setting target of Frontier's
   « operations to that location
   « 3. Clear contents at location "scratchpad.netScript"
   « 4. Set contents of location "scratchpad.netScript" to
   « text contained in netStringStr
   « 5. Reset Frontier operation target to the testConnect
   « script

   new (scriptType,@scratchpad.netScript)
   target.set (@scratchpad.netScript)
   op.wipe()  « Clear current contents of netScript
   op.setLineText (netScriptStr)
   target.clear ()

   « Construct script argument:
   « 1. Delete current contents at Object Database location
   «   "scratchpad.scriptArgument", if any
   « 2. Create new string at location
   « "scratchpad.scriptArgument" to
   « hold parameters transmitted with command to run
   « "Stimuli.PVEP.runStim" script.

   if defined (scratchpad.scriptArgument)
     delete (@scratchpad.scriptArgument)
   new (stringType,@scratchpad.scriptArgument)
   scratchpad.scriptArgument=str

   « Send argument to stimulus computer via
   « NetFrontier.broadcast

   if not NetFrontier.broadcast (@scratchpad.netArgument)
     speaker.beep ()
     delete (@scratchpad.netScript)
     delete (@scratchpad.netArgument)
     return (false)

   « Send script command to stimulus computer via
   « NetFrontier.runScript

   if not NetFrontier.runScript (@scratchpad.netScript,false)
     speaker.beep ()
     delete (@scratchpad.netScript)
     delete (@scratchpad.netArgument)
     return (false)

   « Clean up

   delete (@scratchpad.netScript)
   delete (@scratchpad.netArgument)
   return (true)
```

As in the testConnect script of **Listing 1**, The netRunStim script creates a local string variable to hold the text of the command that will be sent via NetFrontier to Frontier on the stimulus Macintosh. Remember that the purpose of NetFrontier is to direct a copy of Frontier on another computer to execute one of its own scripts. This is what netRunStim will do — instruct Frontier on the stimulus Macintosh to execute a script named runStim contained in its own Object Database. A temporary script (containing the Frontier script to be executed on the stimulus computer) is constructed exactly as was done in **Listing 1**, but in this case the name of the script we are asking to execute on the stimulus computer is "Stimuli.PVEP.runStim". In other words, the runStim script resides in the stimulus computer's Frontier Object Database in its Stimuli.PVEP script table.

The next steps that the netRunStim script take are to send data to the stimulus computer — data that specifies how the stimulus display will appear. This data forms the arguments to the runStim script that will be called on the stimulus computer. As we did above for the script itself, storage for the argument of the script is constructed in the "scratchpad" region of the Frontier Object Database, this time a string variable stored at the "scratchpad.netArgument" location. If a pre-existing argument string is still lingering at that location from a previous execution of this script, it is cleared. Finally, the content of the string "str" — the stimulus parameter list "50 380 8" — is copied directly into the string stored at "scratchpad.netArgument" .

Now we are ready to transmit the script and the script argument to the stimulus computer. This is accomplished in two steps: (1) The NetFrontier broadcast command is first used to send the script argument across the network. If this transmission fails, the user is warned with a beep and all allocated storage is cleared. (2) The copy of Frontier on the stimulus computer is told to execute the script (named Stimuli.PVEP.runStim), using the script arguments just shipped, via the NetFrontier runScript command.

The copy of Frontier running on the stimulus computer must now receive this command along with its argument, decode it, and execute its own runStim script. This script is shown in **Listing 4**.

Listing 4: *runStim* script

runStim

```
« Frontier script within the stimulus computer's Object
« Database for receiving script sent by the recording
« computer in Listing 3.The runStim script extracts the
« script command sent by NetFrontier.runScript and the
« stimulus parameters sent by NetFrontier.broadcast, then
« constructs an Apple event containing the
« stimulus parameters to the stimulus display program.
« Returns true if successful, false otherwise
« © 1997 by Scott B. Steinman. All rights reserved.

« Extract stimulus parameter strings from incoming Frontier
« script from recording computer.
« Place stimulus parameter values into local integer
« variables

local (sPer=long(
        string.nthWord( scratchpad.scriptArgument,1 )))
local (diam=short(
        string.nthWord( scratchpad.scriptArgument,2 )))
local (tPer=long(
        string.nthWord( scratchpad.scriptArgument,3 )))

« Construct Apple event to transmit parameters to stimulus
```

```
« program. This Apple event will instruct the stimulus
« display program to modify the grating stimulus parameters
« and set up the graphical stimulus display. However, the
« stimulus will not be displayed by the stimulus program a
« until 'program ready' sync signal is received across the
« digital I/O lines from the recording computer.

« The arguments to the Frontier "Apple event" command are:
« 1. 'STIM' — the suite or group of Apple events to which
« the stimulus-running Apple event belongs
« 2. 'RnSt' — the identifier for the stimulus-running Apple
« event that tells the stimulus display program to prepare a
« stimulus display
« 3. 'scyc' — the identifier for the parameter that
« specifies the number of grating spatial cycles to display
« 4. sPer — the value of the spatial cycles parameter,
« stored in the Frontier long integer variable sPer created
« above.
« 5. 'diam' — the identifier for the parameter that
« specifies the grating diameter
« 6. diam — the value of the diameter parameter, stored in
« the Frontier short integer variable diam created above.
« 7. 'tcyc' — the identifier for the parameter that
« specifies the number of grating temporal cycles (contrast
« reversals) to display
« 8. tPer — the value of the temporal cycles parameter,
« stored in the Frontier long integer variable tPer created
« above.

if not AppleEvent
(Stimuli.PVEP.id,'STIM','RnSt','scyc',sPer,'diam',diam,
 'tcyc',tPer)
   « If not successful, clean up and warn user with beep

   delete (@scratchpad.scriptArgument)
   speaker.beep ()
   return (false)

« Clean up

delete (@scratchpad.scriptArgument)
return (true)
```


TestTrack
Your Total Bug Tracking Solution

Powerful enough to handle all your bug tracking needs. **Affordable** enough to put on everyone's desktop.

Introducing TestTrack 1.5 and Solo Bug!

Discover the tool more and more of today's top software developers are using to improve the quality of their Mac and Windows applications—**TestTrack**.

- Track bugs, feature requests, customer information, and more.
- Use on Mac or Windows 95/NT platforms.
- Produce concise reports.
- Link engineers, testers, managers, even tech writers.
- Automatically route bugs to team members.
- Save time and improve tech support by giving **Solo Bug**, our stand-alone bug reporter, to your customers (comes free with TestTrack 1.5).
- Track the history of each bug.
- Multiple users, full security!
- Ideal for consultants too!

To order call 513-683-6456

Seapine Software, Inc. 1066 Seapine Ct. Maineville, OH 45039

sales@seapine.com
http://www.seapine.com

Seapine Software

The program first extracts each individual stimulus parameter contained in the script argument from a string in the scratchpad. Local variables are created in the form of named variables of specific types. The first such variable, named sPer, is a long integer to hold the first script argument, the desired grating spatial period. The grating diameter is the second of these arguments, and the grating reversal period is the third. With the stimulus parameters retrieved and separated, they can be packaged into individual Apple event parameters to be transmitted to the counterphase sinewave grating display program. Frontier's Apple event command does this for us. The Apple event command is passed the application signature of the grating stimulus program, stored in the Object Database location Stimuli.PVEP.id. The signature for the C++ sinewave grating display program is 'SinG'. Following this is the sinewave grating display program's Apple event Suite; that is, an identifier for a group of Apple events specific to that program. We use the Apple event Suite identifier 'STIM'. The following identifier is that of the sinewave grating display program's runStimulus Apple event. This identifier allows the display program to recognize the command to display a sinewave grating stimulus. All that remains is to pass the individual Apple event parameters that define the desired appearance of the sinewave. These form the three remaining pairs of arguments to the Apple event command. Each pair consists of an Apple event parameter identifier for the stimulus parameter to be used when the stimulus program decodes the Apple event, and the value of the parameter. For example, the grating spatial period is given the Apple event parameter identifier 'scyc' and the value of the Frontier sPer variable follows it.

When the Frontier Apple event command is executed, Frontier transmits a raw Apple event that will be received by the stimulus display program. This forms the last step in the chain of message-passing from LabVIEW, to Frontier on the recording computer, to the second copy of Frontier on the stimulus computer, to the stimulus display program. It is now the job of that program to decipher the incoming Apple event to determine what the stimulus display program has been asked to do by LabVIEW. In other words, the display program need not "understand" the scripting code output by LabVIEW, nor the scripts transmitted by Frontier. Such code is difficult to decipher by C++ programs. Rather, it need only understand very basic, low-level Apple events, which require a minimum of programming to decode. Our Frontier programming has allowed us to simplify our programming task for each stimulus display program to the point that each program needs only be a small skeleton program written in C++ consisting solely of Apple event-handling and stimulus generation graphics code.

The Apple event handler called when the 'RnSt' Apple event is received is shown in **Listing 5**. Before this code is examined, the code to set up Apple event handling must be explained. Prior

to entering the event loop of the stimulus display program, the means by which the program receives high-level events like Apple events, the Apple event handler must be installed with a call to the InitAEStuff function of **Listing 5**.

Listing 5: AppleEvents.cp

AppleEvents.cp

```
// C++ code in the stimulus display program that receives and
// handles the Apple event transmitted by the runStim
// Frontier script of Listing 4. This Apple event handler
// parses the stimulus parameters from the Apple event, then
// calls the DoAETrial function to display the stimulus
// (a sinewave grating in this case).
// Each stimulus parameter has its own Apple event keyword
// identifier to allow easy extraction from the Apple event.
// © 1997 by Scott B. Steinman. All rights reserved.

#include <AppleEvents.h>
#include <EPPC.h>
#include <GestaltEqu.h>
#include <PPCToolbox.h>
#include <Processes.h>

static ProcessSerialNumber gPSN;

void InitAEStuff( void )
{
  static AEinstalls HandlersToInstall[] =  {
    // Required Apple events (not shown),
    // plus our custom Apple event.
      { 'STIM', 'RnSt',
        (AEEventHandlerUPP) AERunStimulusHandler }
    };
    OSErr  aevtErr = noErr;
    long   aLong = 0;
    Boolean  gHasAppleEvents = false;

    // Get process serial number of this program.
    aevtErr = GetCurrentProcess( &gPSN );

    // Check machine for ability to handle Apple events.
    // If not present (ie, not System 7.0 or above), exit.
    gHasAppleEvents =
      (Gestalt( gestaltAppleEventsAttr, &aLong ) == noErr);

    // Installs our Apple event Handler. Whenever an Apple
    // event is received and we call AEProcessEvent, the
    // Apple event manager will check our list of handlers
    // and dispatch to our custom Apple event handler, if it
    // exists.

    if ( gHasAppleEvents ) {
    // Required Apple events would also be installed here...
      aevtErr = AEInstallEventHandler( 'STIM', 'RnSt',
        (AEEventHandlerUPP) AERunStimulusHandler, 0, false );
      if ( aevtErr )
        ExitToShell();  // Just abort program if error occurs

      AESetInteractionAllowed( gInteractNow );
    } else
      ExitToShell();
}

// Apple event handler called when Apple event with
// identifier of 'RnSt' in suite 'STIM' is received.

pascal OSErr
AERunStimulusHandler( AppleEvent *messagein,
                      AppleEvent *reply, long /* refIn */ )
{
  DescType returnedType;
  Size     actualSize;
  OSErr    error;

  // Bring stimulus display window to front
  error = SetFrontProcess( &gPSN );
  error = AEInteractWithUser( kAEDefaultTimeout, 0,
                              (AEIdleUPP) idleProc );
  if (error != errAENoUserInteraction) {

    // Extract stimulus parameters from incoming 'RnSt'
    // Apple event

    // Stimulus spatial period
    error = AEGetParamPtr( messagein, (AEKeyword) 'scyc',
      typeLongInteger, &returnedType, (Ptr) &spatialPeriod,
      sizeof( spatialPeriod ), &actualSize );
    if (error) return( -1111 );  // Our own unique error code

    // Stimulus grating diameter
    error = AEGetParamPtr( messagein, (AEKeyword) 'diam',
      typeShortInteger,  &returnedType, (Ptr) &diameter,
      sizeof( diameter ), &actualSize );
    if (error) return( -2222 );

    // Stimulus temporal period
    error = AEGetParamPtr( messagein, (AEKeyword) 'tcyc',
      typeLongInteger,  &returnedType, (Ptr) &cycleVBlanks,
      sizeof( cycleVBlanks ), &actualSize );
    if (error) return( -3333 );

    DoAETrial();
  }
  else
    return( -1 );

  return( noErr );
}

void DoAETrial( void )
{
  // PlayGrating waits for digital signal before displaying
  // sinewave grating visual stimulus.
  PlayGrating();
}
```

InitAEStuff first determines the process serial number of this stimulus display program. The process serial number is a unique identifier for each program or task currently executing on the computer. The GetCurrentProcess Toolbox routine retrieves this serial number for us, and stores it in the global variable gPSN. After confirming with the Gestalt Manager that this machine does in fact support Apple events, we install the Apple event handlers. Here we show only the example handler and omit for the sake of brevity the four required Apple events for opening a program or program file, printing and quitting a program. Finally, we call AESetInteractionAllowed to set the program interaction mode, that is, how the user is permitted to interact with this Apple event "slave" program — is the program run solely in the background, can it receive mouse clicks from the program user, and must the calling program be on the same computer to allow such interaction? We permit user interaction from calling programs on any computer, as most programs will.

Now we can examine the Apple event handler code itself. The first act of the example AERunStimulusHandler handler is to bring the stimulus graphics display window to the forefront by calling SetFrontProcess with the process serial number of the stimulus program, then we initiate program interaction if it was permitted. At this point, we can retrieve the stimulus parameters contained in the Apple event that will determine the appearance and behavior of the stimulus display. Each parameter is extracted from the Apple event with AEGetParamPtr, which is passed as two of its parameters an identifier or Apple event keyword that determines which parameter is retrieved, and the type of variable the parameter is. In the case of this sample code, the stimulus is

a spatial sinewave grating. The first stimulus parameter stored in the Apple event to be retrieved is the spatial period of the sinewave. This parameter's keyword is 'scyc' and its type is a long integer. AEGetParamPtr stores the retrived parameter in the stimulus program's long integer variable spatialPeriod. If any error occurs in this process, the process is aborted. A similar sequence of steps is used to retrieve the grating diameter and counterphase reversal temporal period. If each parameter is extracted successfully, the DoAETrial routine is entered, which is responsible for displaying the grating stimulus. DoAETrial sends a sync signal via the digital I/O line (noted by the Boolean variable stimEnabled) to the recording computer to signal that the stimulus is ready and recording may begin.

One aspect of this inter-program control that has not yet been discussed is the launching and quitting of the stimulus display program prior to and following data acquisition. Frontier simplifies these tasks as well. NetFrontier can instruct the copy of Frontier on the stimulus computer to execute two other Frontier commands: The first is Frontier's launch command, which sends an Apple event via the Finder to launch a program. The form of this command that we use is launch.usingID(Stimuli.PVEP.id). This command finds the application whose signature is stored in a variable called "id" in the Object Database table PVEP within the Stimulus script suite. The second command, to be used after data collection, is Frontier's quit command. This command takes the form core.quit(Stimuli.PVEP.id, "no"), where the second argument is a string stating whether or not a data file is to be saved when the requested program is quit.

The example code demonstrates that Frontier can be used to handle all aspects of the inter-program communication that is at the core of the laboratory electrophysiology software. Frontier launches the stimulus generation programs, signals each to display stimuli with specific stimulus parameters, then forces them to quit when we no longer need them. The sole purpose of the digital I/O lines is to ensure the tight time-locking of stimulus display and data recording.

Conclusions

In the construction of software systems, complex decisions regarding individual aspects of the system design must be made that have serious implications for the design of the remainder of the system. In the present case, the overwhelming need for program execution speed dictated the choice of a dual-computer system and the method of communication between the computers. Such a system could not have been constructed easily without combining the strengths of LabVIEW, Frontier, and C++. The use of LabVIEW shortened the program development cycle considerably. In addition, inter-computer communication of the speed and complexity used here would not have been achieved easily using C++ without the addition of Frontier.

Despite the complexity of this system, the software is easy to operate by the clinician due to the intuitive user interface imposed by LabVIEW's instrument panel paradigm and the operation of all tests from a single program on the recording computer. More importantly, the system design does not sacrifice modularity and expandability. Adding a new test requires adding only a small skeleton C++ stimulus display program, a few LabVIEW VIs to select stimulus parameters, send scripts and record data, and a few Frontier scripts. These code modules are very similar in each of the recording, stimulus and intercommunication programs, in great deal due to a high degree of code reuse in LabVIEW, Frontier and C++ libraries for handling Apple events and generating animated graphics. Much of the task of creating a new set of tests simply involves duplicating, then modifying, existing code.

In the future, it may be possible to remove some of the complexity of the system once it is possible to perform all of the stimulus display and recording tasks on a single dual-monitor dual-processor computer. The inter-program communication could then be restricted to one machine, eliminating the need for NetFrontier — scripts could be sent from LabVIEW to Frontier and directly translated into Apple events to be received by the stimulus program. Frontier can use the fast Component Manager of Power Macintoshes for Apple event transmission within a single machine, which increases the speed of inter-program communication even more. However, even with the present single-processor technology, we have demonstrated that the creation of powerful, flexible, real-time software is facilitated when it makes use of innovations found on the Macintosh computer. The same programming principles outlined in this paper may be applied to a wide range of applications.

Bibliography and References

- Apple Computer Company, *Inside Macintosh: Inter-program Communication*, Addison-Wesley.
- Baro, John A. and Hughes, Howard C. *"The Display And Animation Of Full-Color Images In Real Time On The Macintosh Computer"*. Behavioral Research Methods, Instruments and Computers, 23 (1991), pp. 537-545.
- Johnson, Gary W. *"LabVIEW Graphical Programming: Practical Applications In Instrumentation And Control"*. (1994), New York: MacGraw-Hill.
- Steinman, Scott B. *"Simple Real-Time Color Frame Animation"*. MacTech, 9:9 (September 1993), pp. 21-35.
- Steinman, Scott B. *"Extendable Real-Time Simultaneous Data Acquisition And Stimulus Generation On The Macintosh Computer"*. Behavioral Research Methods, Instruments and Computers, In Press-.
- Steinman, Scott B. and Carver, Kevin. *"Visual Programming with Prograph CPX™"*. (1995), Greenwich, Connecticut: Manning Publications / Prentice-Hall.
- Steinman, Scott B. and Nawrot, Mark. *"Real-Time Color Frame Animation For Visual Psychophysics On The Macintosh Computer"*. Behavioral Research Methods, Instruments and Computers, 24 (1992), pp. 439-452. MT

Want to suggest an article for the magazine? Send your suggestion to <mailto:editorial@mactech.com>

by Kee Nethery & Kagi clients, Berkeley California USA

Distributing Software on the Internet

Advice from people who have learned the hard way

Introduction

All programmers dream of retiring on money earned from a program written in their spare time. They know the coding part and they can envision the retired part, it's the steps between where "magic happens" that are just a bit fuzzy in the dream.

Magic Happens

Absolutely the most successful products distributed on the Internet are those that: are good, are usable by the largest number of people, are easy for people to use on a trial basis, and have a compelling reason for people to pay. No one is born knowing how to create such a product or getting people to pay you. While you spend lots of time learning all these skills...

Do Not Quit Your Day Job

Even if your product is wildly successful and you are earning many times more than your day job, keep in mind that your massively successful product could become obsolete tomorrow and your income could suddenly cease. You and your family will sleep easier if you keep a day job that pays the bills and puts food on your table. Consider reducing your monthly overhead by paying for your home, paying off all your loans, saving money for future large expenses, etc. As you reduce your monthly expenses, you can safely reduce from a full time job to a part time job. When your monthly expenses are so small that you can earn that amount each month by working a day or two as a consultant, you'll know that the day job is not required. Until then, always have a way to pay your bills because the chances are that your first product will need lots of tweaking over a long period of time before it has a chance of being successful. You will not be able to program if you are worried about finances.

The above advice has nothing to do with creating a good product. So how to create a good product. Telling someone to write a good product is like telling them to...

Buy Low and Sell High

You should write a product that everyone will want to purchase. Well, duh. :-) Write software that others think is better than the competition. One easy way to be better than the competition is to create a product that leverages some expertise or passion that you already have. If you spend idle time pondering some subject area, chances are good that you might be the best informed programmer in that subject area.

You should be able to do a very good job of writing software in your subject area because you know it so well. Within your area of passion and expertise there will be a wide range of products that you could create. Look at the existing products and envision one such that...

Everyone Will Want It

Before you write one line of code, find twenty or more people who would want your type of product and ask them what they would want it to do. If you cannot find enough people to give you suggestions, either there is not a big enough market or you need to first figure out how to locate these people. These people will be your beta testers. Once you start to get suggestions from lots of people, consider if this market has money to pay for this kind of software and if what they want will sell to lots of people or just a very tiny segment of these people.

Within your target market, imagine all the people who use your chosen computing platform (for the purposes of this article, Macintosh users). Imagine what percentage of these people share your passion. This will be a small subset, lets say 1 out of 100 of the overall users of that platform, 1%. Within this

Kee Nethery operates Kagi, a payment processing service for people who sell products on the internet. This article is composed mostly of expertise contributed by the programmers who produce products sold by Kagi. They and their products are listed on the web site at <http://www.kagi.com>.

mall subset, you have a variety of product types; single user, nulti-user, programming tools, etc.

Software that appeals to your entire market might sell to 1 ut of every 10 of the 1% = 0.1%. Server or multi-user software hat is shared by many people might sell to 1 out of 20 of the 0.1% group that would purchase the single user version = 0.005%. Programmer and developer toolkits might be used by 1 out of 100 of the 0.005% group that would purchase a server = 0.00005%.

When you consider writing a programming toolkit, a server, or a single user product, keep in mind the size of the market and price your product accordingly.

Develop a product that you are expert in AND keep in mind what percentage of the overall potential market your product will sell to. A product that deals with controlling telescopes might get 100% of that market (sell a copy to everyone) and still make very little money as compared to a screen saver that sells to less than 1% of it's entire market.

Writing the Software

Your product has to be competitive. Learn about your competition. Buy their products, use them and understand what is good and bad about them. Your product must be the best or you should find a different market. Keep in mind that best does not mean biggest and most feature rich, it means that your testers like your product better than all the others.

Don't be afraid of big name competition, you can beat the big companies. Their teams of salaried programmers are like huge oil tankers that take miles to turn or stop. You are the wind surfer who knows all, sees all, and can change direction in seconds. You are absolutely the most efficient programming team that is possible, one programmer with the complete program in your head and a passion to do it right.

One of the biggest factors influencing Internet software sales is not how well the program performs its function, but how solid it looks and feels. The user interface is critical. A rock solid product with a shabby interface will be perceived as inferior to a buggy product with a pretty interface. People do judge a book by its cover so find someone who can give your product an excellent user interface.

The First One Is Free

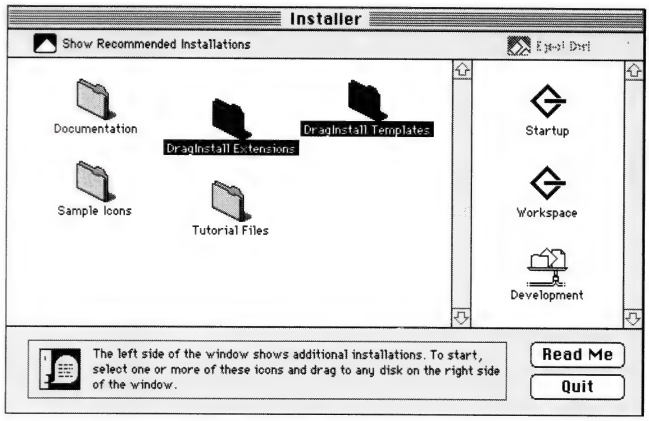
This is the key to Internet software distribution. You want to provide enough functionality for a long enough period of time so that people become accustomed to your product and not having it is not an option for them. The easiest way to do this is to provide a fully functional product that is easy to use and does things that people want. Of course this is in direct conflict with...

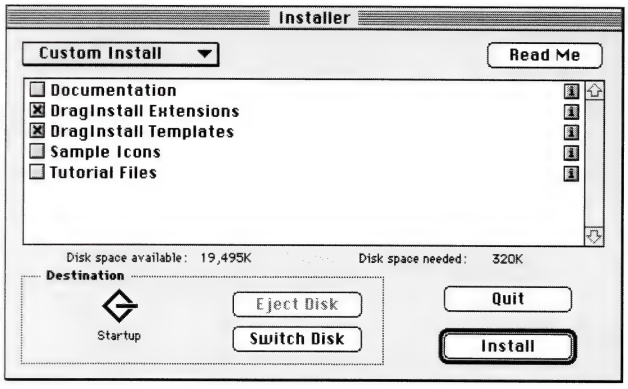
Asking People To Pay

For any product distributed on the Internet there is a slider that goes from "Full Functionality and Payment Optional" to "No Functionality until Payment Received." Finding the optimal balance between these two extremes is not easy.

You want to provide enough functionality for a long enough period of time so that people become addicted to your product

Guess which installer was built by DragInstall®





Guess again.

If you guessed that the first installer was built by DragInstall, you're only half right. Now with DragInstall 3.0, you can build *both* of these installers from a *single* script file!

For more information about DragInstall and a free demo, visit our web site at **http://www.sauers.com/draginstall** or give us a call at **1-800-890-9880**.

and are enticed to pay. How you entice them to pay depends entirely upon what kind of product you have. Spend time brainstorming with friends and your beta testers on where to draw the line between the two extremes. Push them to pay but do not push so much that they stop using the product. These payment request schemes might give you ideas for your product:

Desktop Appearance Manager

Software that beautifies the desktop. This software pops up a message maybe once or twice a day that asks for a payment. The message it delivers is humorous and sincere and the unpredictable appearance of the message causes people to want to have it stop interrupting them. The software is fully functional and it is not essential to anyone's business. When they pay, they are mainly paying for the code number that causes the annoying dialog to cease.

Game

A network based multi-user action game. Imagine an annoying dialog that pops up during every 5 games or so that comes up at a critical moment and requires that them read it to figure out how to make it go away. During that time, the other users will most likely kill their character and they will lose the game. Most people will pay to keep that from happening.

Graphical Utility

If the utility does graphical manipulations, you could put an advertisement along the edge of the resulting image. When they pay the advertisement goes away.

Network Utility

For network utilities, you could require that they restart the program after every ten actions, you could randomly hide some results, you could limit them to at most two simultaneous actions, you could timer the software so that after 30 days it starts to become annoying. It depends upon the product.

Word Processor

Limit the size of the document that can be produced, or require that they print one page at a time.

Network Server

Starting after a month of use, every couple of weeks at 2 am pop up a "Please Pay" dialog that stops your server from functioning until they click on it. If your server software is critical to their users, having users complain early in the morning will probably get you a registration fee.

Operating System Extensions without an Interface

For a faceless piece of code, in many cases the only thing that can be done is to put up a payment request on startup. If that is all you can do, that is what you do.

Convincing People to Pay

There is a big difference between asking for a payment and convincing someone to pay. When you ask them to pay you want to convince them that it is worth their while to pay. You are creating an advertisement and how you craft it is very important.

Of course your splash screen (the first one seen when launching your product) should ask for a payment if the software has not yet been paid for. But, do not just put your payment request message at startup. People will treat it as a normal part of using your software and they will ignore it. Ask again sometime during the day and hopefully you will remind them while they are trying to make a good impression on someone.

Do not dismiss the payment request message through a simple keyboard button press. Swap the buttons around. Make them read the buttons and select the correct one with their mouse. If they always select the same button to dismiss your payment request, not paying will become a habit. You could also have them do a simple math problem like adding two numbers together to dismiss the dialog. Eventually they will decide that the small fee you are requesting is worth it just to not have to have the payment request interfere with their normal operations. Use TV commercials as your guide. You are the sponsor of this product and making them react to your commercial is part of the game.

Do keep track of user metrics such as; when they first installed the product, how much they have used it, how much time you estimate they have saved or in the case of games, how much time was spent. Give them reasons to pay based upon their usage of the product. Vary your payment request message based upon their usage history.

Some people provide very short trial periods for their software. Few people will install some software and then devote the next two days of their lives to a massive testing of your software. Most people will install it, launch it, and see what it does then not use it for another month or two until the need for your software presents itself to them. Do not use short trial periods. Instead consider counting the actual hours used. If they have been using it actively for 10 hours over a period of a year, chances are they have a pretty good idea what it does and whether they should purchase it or not.

Do not use a specific calendar date to determine when the software should cease to function. We still receive payments for 3 year old software that people have just found on a floppy and are just now trying. If you kill the software on a specific calendar date, you will lose sales. Worse, after that date you will start to receive an increasing number of support questions from people who have not yet paid.

Protection Schemes

Once they pay, you must acknowledge their payment. Some people develop massively complex password schemes to prevent multiple users from using the same password key. Be careful, you don't want to prevent a paid user from moving their software to their new hard disk, etc. If you create a very complex protection scheme, you will spend lots of time just helping them move software from one disk to another. Paid users will dislike the hassles created by the protection and they will be less likely to recommend your product to their friends.

Extremely secure protection schemes can prevent people from becoming addicted to your product. Do not overprotect

your software. Some of the most successful software has the least restrictive protection. In my opinion, you should accept that there will be theft and primarily focus on providing the best product that you can.

The following are some samples of the various protection schemes I have seen on successful software.

None

Clicking on the "I've Paid" checkbox in the preferences window causes the "Please Pay" dialog to cease to appear.

Secret Move

Some software has a field where you can enter the registration code but nothing entered into that field will ever unlock the software. Instead there is some secret move that unlocks the software. For example: click the red shoes three times and then type OZ.

Secret Code: Universal

There might be one secret code that you give to everyone who pays. They all get the same secret code.

Secret Code: Random

The secret code might be randomly constructed where specific characters in the code are derived from the other characters so that your software can determine that the code was probably generated by you. Otherwise the characters are random.

Registration Code: User Data

It is very common to derive the registration code from information provided by the user and then to display that user information in the software so that their information gets passed to anyone to whom they give out their code.

Unlocked Version

When someone pays, they are sent or given instructions on how to download the unlocked version of the software.

Caveats

Do not tie your software to someone's hard drive parameters. People reformat their hard drives, they upgrade to larger disks, sell their machines, etc. If you tie your protection to hard drive parameters, you will do lots of support for paid users every time there are changes to their hard drive.

Do not threaten to erase their hard drive if they are using a known pirated registration code because you will get blamed for the next bad thing to befall their computer. Do not even joke about erasing their hard drive.

Pirates

If your software has any value, its protection scheme will get cracked and registration codes that work will get posted on the internet. Keep track of registration codes that have been posted to the internet and in future revisions, disallow these registration numbers.

Do check for registration code validity in several places in your code and only enable some of these various checks on, for example, odd weeks or even months. If someone publishes a patch for your software, it will work until the next month and then it will cease to work. The cracker will have moved on to other programs and everyone who was using the pirated version of your code will find that it ceases to function.

If you are worried about pirates posting your registration codes to the internet, and you know where such codes get posted, before you release your product, you might wish to post some fake registration codes that cause the program to appear to be unlocked. Why should a pirate crack your software when some other pirate (you) has already done so? After some period of time, the software can then revert back to being locked.

Time to Ship

When you are completely finished with all your documentation and bug fixes and the product is bundled up and ready to upload to the Internet, send it to your beta testers and have them beat on it for a couple of weeks. Repeat this cycle of beta testing and bug fixes until all the beta testers report no problems. Only after they say it is ready to go should you release it to the world.

Pricing

For the software that Kagi handles, US$ 10 seems to be on the low end and US$ 80 seems to be on the high end for single user software. The average seems to be around $18. The best price is the one that brings in the most revenue and choosing that price is always a balance between selling millions of copies at a one dollar each and selling only one copy at a million dollars. Ask your beta testers for their opinion. In the final analysis, your software is only worth what others are willing to pay for it.

A program that does something simple and is not critical to the user should be priced less than a program that is essential to a user's productivity. People will pay more for a unique product with no competition, for a useful product, and for a product that is just way cool to have and enjoy.

A low price is only essential when someone has to buy it before they can try it. If someone can try your product before they buy it, the question they ask themselves is whether it is worth the money to remove the annoyance, not whether the price is too high. For many people the price is not an issue, it is the hassle of paying that delays their payment. Once you set the price, changing it can create problems so give it some thought.

How to do a price increase

It is easy to lower your product price, it is difficult to raise it. The best way to raise your product price is to create a new and improved version of your software that justifies the price increase. Secondly, you should give it a slightly different name, for example you might go from MyProgram to MyProgram Pro or MyProgram Version 2.

For a smooth price increase, you should have three prices, MyProgram at the old price, MyProgram Pro at the new price, and Upgrade to MyProgram Pro at the difference between the two prices.

The registration codes (if any) that you issued for MyProgram should not work by themselves for MyProgram Pro. Have new registration codes for MyProgram Pro. Create two types of MyProgram Pro registration codes, one that can be used by itself and one that requires the earlier MyProgram registration code to be complete. For the people that are upgrading, just give them the upgrade portion of the code.

There are other ways to do price increases but this seems to work pretty well for most people. Vastly improved product justifies the price increase and giving it a slightly different name tends to eliminate the confusion that people might have concerning which price to pay.

Sales Results

You are now at the point in the product cycle where you have a huge advantage over big mega-companies. If your product is not providing the financial success you expect, run experiments to find out why and then try corrections. Perhaps people are not getting exposed to your product and thus they are not becoming addicted. Perhaps they love it but are not paying for it. Perhaps there is better software available, or the price is too high.

Give yourself time to identify whether it is your product or your marketing of it that needs to be revised. Try things and see what works.

Exposure

Part of what you must do is to get people who should want your software to try it. If they cannot use it immediately without reading the manual, people who would otherwise love it will never be really exposed to it - make it usable. If people do not download it, you need to do a better job of reaching your target market and keeping them exposed to your product. Issue revisions and explain in the revision notes that you post on mailing lists and in news groups what problem the revision solves so that others with that specific problem will give it a try.

Keep in mind the adage that "Any exposure is good exposure". Several software authors have indicated that their sales went up drastically after bad software reviews in major magazines. Don't be afraid to get a negative review.

Slow Payments

Perhaps your users are weighing the annoyance of your payment requests with the cost of the program and are deciding that the annoyance is more cost effective. You can lower the cost to below the annoyance level or you can increase the annoyance above the cost. Experiment and see if either works better.

Competition

Your software must be better than your competition by some noticeable way. Find out what your competition cannot do that users want and provide that. This does not mean compete feature by feature, this means find a group of people who are unsatisfied with your competition and make them happy.

Price Too High

I have never heard someone indicate that lowering the price of their product by half doubled their sales. Only lower the price if that is the only objection for people. When they say it is the wrong color and it is too big and it is too expensive, ignore the too expensive part and concentrate on the other complaints.

Summary

Write a good product in some area where you have expertise. Before you write it, make sure that lots of people would want to use it. Make it easy for them to try it. Bug them just enough to get them to pay but not so much that they stop using your software. Do not quit your day job. MT

Want to suggest an article for the magazine? Send your suggestion to <mailto:editorial@mactech.com>

y Dave Mark, ©1997 by Metrowerks, Inc., all rights reserved.

CodeWarrior Rhapsody Update, Part 2 and a Quick Look at WarriorWorld

Last month's Factory Floor column featured the first half of an interview with the CodeWarrior Rhapsody team. This month, we'll finish off this interview, starting with a look at the yellow box debugging plans. We'll also take a quick look at a cool new web site, known as WarriorWorld...

Berardino E. Baratta *is Vice President, Research and Development for Metrowerks Corp. With the Metrowerks R&D department quickly approaching 100, he has little time left for doing actual coding but he does still sneak it in (mostly by working on it late at night, or early in the morning, depending on how you look at it). Most recently having done compiler and linker modifications for PalmPilot Release 3. He also tries to spend as much time as possible with his wife and young son.*

Since starting at Metrowerks in early 1996, Lawrence You has been helping shape the CodeWarrior debugging architecture for non-Mac OS targets, including Rhapsody. He has been a longtime Macintosh developer, also working for Apple and Taligent.

David Hempling *is the CodeWarrior Latitude Technical Lead at Metrowerks, where he is bringing the Latitude Technology to Rhapsody as well, opening new doors for Metrowerks existing applications. David co-founded Quorum Software Systems in 1989 where Latitude was born and has nurtured this unique porting technology as The Latitude Group's President from 1994 to 1996.*

Dave: Will MetroNub exist in the yellow box? What are the yellow box debugging plans?

Lawrence You: From the point of view of the CodeWarrior developer, the debugging environment will be very similar, if not identical, to the environment hosted on other platforms. We will know that we did our job well if today's CodeWarrior user cranks up their copy of MW Debug on Rhapsody and says "hey, this looks exactly the same as what I'm used to." There is a growing emphasis on cross-platform development, making ease-of-use and consistency very important in debugging environments.

But, why stop at developing and debugging on only one platform at a time? A developer's needs are different than the customer. Why should a developer be required to use a different platform and often different development tools for every target they develop for? If they are comfortable using an environment on Rhapsody (or MacOS or WindowsNT) and want to use it to cross develop all their versions in one place, why should they have to learn how to use a new debugger? Consider just the number of Yellow Box implementations alone that a developer will have to use: to start, there will be native Yellow Box in PowerPC, native Yellow Box on Intel, and the Yellow Box API on Mac OS.

Considering all of the possibilities, our main goal is to evolve our existing debugging environment into one that can be used for many targets at the same time. Just as we've recognized that our CodeWarrior IDE needed to have the ability to create multiple "targets" which can be used to create fat or cross-developed applications, we also recognize that we have to provide a solution that will let a CodeWarrior user debug for those different targets.

You can see evidence of this strategy already. Just to give you an idea of the extent we believe in this, our Macintosh and Windows debuggers already support C/C++ on MacOS/68K, MacOS/PowerPC, Windows 95/NT/Intel, Sony PlayStation/MIPS, Motorola PowerPC EABI (embedded)/MPC821 and MPC860, PalmOS/68K, OS9/PowerPC not to mention Java and Pascal on

the desktop platforms. So along those lines, the Rhapsody targeted debugger will "just" be another plugin; to the CodeWarrior user, MW Debug will look just about the same.

But, there are differences which we'll be addressing. Objective C is different than C and C++. Object file and symbolic information formats are different. MetroNub, which is just a low-level Macintosh INIT, will be replaced by a server. Multitasking and threading will be more prevelant; multiprocessor hardware will be less forgiving of errors in concurrent programs. These are just a few examples.

The Objective-C runtime will require we change the debugger to allow the user to inspect objects and polymorphic functions. Also, the exception handling mechanism is different from C++ but the user will still have the ability to catch exceptions with the debugger. We should also be able to maintain our feature of being able to debug across virtual machine boundaries, like you see with programs which use mixed 68000, PowerPC or JavaVM code.

The linker will generate Mach-O output for PowerPC and Intel, which is different from PEF and XCOFF files. Source-level debugging information will go into a separate debugging "section" of the file, although we may also give the user an option to split the symbolic info into a separate file, kind of like ".SYM"/".xSYM". We have been using DWARF for our MIPS and PowerPC embedded compilers and debuggers and have found it to be a more neutral and expressive debugging information format, not to mention that it is a standard. So we'll be using that instead of the Apple SYM format. To ensure a high level of interoperability, we're also working with Apple to help ensure that files created and used by gnu tools (gcc and gdb) can be used in a mix-and-match fashion.

MetroNub is fine for MacOS debugging, but since Mach differs so dramatically from MacOS, we're moving towards a debugging client/server architecture that can be used on not just Rhapsody, but also for other future systems. Some notable differences are that there is the notion of a single user using a host on MacOS. The Mach tasking model allows multiple users, and the new server will allow for that. The new debugger server can be used locally and remotely with little difference except for responsiveness, opening up possibilities for WAN-based graphical debugging.

Driver developers should find remote debugging with a microkernel useful. It's not clear to me yet what low-level code you won't be able to debug locally (or in some cases remotely), but it seems likely many drivers that were impossible to debug on a Mac with CodeWarrior should look similar to applications on Mach. At the very least, we'll make a strong effort to not preclude driver debugging.

Like other modern operating systems, Mach will allo multithreaded programs to run truly concurrently o mulitprocessor machines. This opens up some ne opportunities and exposes errors. Many developers have see "Heisenbug" effects (even without MP machines), whereby debugger becomes intrusive, ever so-slightly, changing th events in a program. This is an effect that, in some cases, impossible to avoid. Most debuggers on multithreaded system including gdb, allow the user to examine threads and s breakpoints, but we'll work hard to make sure the user has th most flexibility to debug concurrent programs on MP hardwar

Dave: How does PowerPlant play in all this?

Berardino Baratta: We have a large investment in PowerPlan and don't plan on throwing this away, but we are no planning on porting PowerPlant to Rhapsody as a nativ framework. These are many reasons for this but the simples one is that PowerPlant was designed from the ground up t be a Mac OS framework, and as such would require a complete rewrite in order to be efficient under Rhapsody meaning that users couldn't take advantage of thei investment in PowerPlant without rewriting their code. We feel that if users are going to be rewriting their code, ther they should target the Yellow Box API directly and take advantage of that API's object oriented design from the start.

We are still planning on providing our users with a path t Yellow Box through the use of our Latitude porting library. We ourselves are relying on this path in order to port our IDE native to Rhapsody. The combination of Latitude and PowerPlant allows users to maintain their current design and yet bypass both PowerPlant and Latitude in order to directly access the Yellow Box APIs in order to implement native functionality, thereby taking advantage of features that are only present in the Yellow Box.

Dave: What's the timeline for all this?

Berardino Baratta: As I write this, we've just recieved a version of Rhapsody on PowerPC, and don't know yet how stable this version is. That is one of the largest variables in our being able to deliver the above mentioned tools in our CodeWarrior Professional Release 2 delivery which is supposed to ship on October 15th. We have full support from Apple engineers so we feel that if it can be done, we will work together to make sure that it is done.

We are not putting all our eggs into the Rhapsody native basket though, in that we are planning on implementing cross development support from MacOS as a first step in our plan. We don't see any road blocks in that plan, so at the very least users will have access to these cross compilers, linker and debugger in Pro2 and we'll then invite users to participate in our beta testing program in order to access our native toolset.

This is only Plan B, as I speak, we're still on track for Plan A.

The Pro 2 release of Rhapsody tools, will be of prerelease quality, but we'll patch them over time, in order to ship final releases of the tools, around the same time that Apple goes final on their first public release of Rhapsody. Very similar to our strategy in 1994, during the transition from 68K to PowerPC architecture for MacOS. During the interim, we plan to allow our users to work native on Rhapsody from Day One of their receipt of the first Developer release of the new Operating System.

Dave: By the time this issue hits the streets, developers will have had a chance to play with both the first developer's release of Rhapsody as well as a new version of Latitude for that release. What kinds of things will people be able to do with this new version of Latitude?

Dave Hempling: Our development of Latitude for Rhapsody has gone extremely well. We prepared for Rhapsody by doing initial development on the OpenStep 4.2 release on Intel while we awaited Apple's PowerPC Rhapsody Developer Release. All of Latitude's File and Resource Manager functionality was made endian neutral at that time so that we could open, read, and write Mac Resource Forks in OpenStep. Once the basic Graphical User Interface components were coded and in place, we were able to begin interpreting WIND, DITL, MENU, CNTL, and other resources and put up dialog boxes.

With working windows in place, we had canvases to accept drawing operations. All of our basic Quickdraw drawing, including Region support and some complex CopyBits operations like hilighting and patterned brush drawing have been mapped to DPS operations. Since color dithering is a feature of DPS, we didn't have to manually do it ourselves — which was a welcome change from Latitude's X version.

All of these features have made their way smoothly to Apple's PowerPC Rhapsody Developer Release. We're successfully mapping windows, controls, menus, and graphical and textual rendering to Rhapsody's GUI and DPS. Some of the Latitude implementations may not be as efficient as we'd like but I know we'll have these cleaned up once Apple releases the Premiere Rhapsody release slated for January.

Developers experimenting the Latitude now should be able to access their resources, put up their windows and dialogs, and see a majority of their application's features working well. Features developers will see missing are those that hadn't made it into the Apple Rhapsody Developer release such as Blue/Yellow Box file sharing and AppleEvents.

One void I know developers will see now is a means to get Mac files into their Rhapsody environments. Simply ftp-ing won't do since apps depend on the Mac File Manager to report various Finder attributes, like file types and owners, and this information is lost by simple ftp. Latitude's File Manager will report all this info if it is included with the files in one of several popular formats that encode Finder information into the file, including AppleDouble, AppleSingle, EtherShare, K-AShare, and uShare. As it happens, Jeff Matthews' Fetch 3.0.1 can write AppleSingle files. Until Helios brings EtherShare to Rhapsody, as they've promised, Fetch 3.0.1 is the only way I know at this time to safely transfer Mac files directly to Rhapsody so that they are fully understood by apps ported with Latitude.

This birth of Rhapsody is much like other new UNIX operating system releases that I've witnessed over the years. Not so much cutting edge but rather bleeding edge development. While the Apple engineers rework and fine tune their new operating system, developers like us are attempting to bring out the best tools possible under constantly shifting conditions. Latitude has been through this kind of development before when operating systems like IRIX 4.0 and Solaris 1 were in early releases. As the Rhapsody releases become more stable, so will Latitude's support.

Check Out WarriorWorld

If you've spent any amount of time teaching yourself PowerPlant, chances are you've heard of the PowerPlant Dream Team, a group of folks dedicated to learning about PowerPlant and interacting and collaborating with each other online.

The founders of the PowerPlant Dream Team created a non-profit organization called WarriorWorld, the official CodeWarrior users' group. Now, WarriorWorld has a new home on the web at <http://www.codewarrior.org/>.

WarriorWorld is ramping up now and, by the time you read this, the site should be complete, offering a range of technical info and services. According to the WarriorWorld FAQ, the site will soon offer a series of free courses in Java, C++, and PowerPlant. Each course will organize a corresponding Dream Team to make the learning process more effective.

To me, this site has huge potential. Check it out... MT

Want to know what products are available for MacOS development? Check out Developer Depot™ <http://www.devdepot.com>

by Jay Van Vark

eCommerce and the Security Myth

The real security issues of eCommerce

eCommerce in Today's Market

Business is done with many communication technologies today, walk-in retail, mail-order phone, mail-order fax etc. The Web and the Internet are just one another communication medium with its own benefits and disadvantages. The cost for a business to have a world wide presence is the lowest in history with the World Wide Web. Budgets of the 1980's would have listed at least $100,000 per month in expenses to have a business handling international customers 24 hours a day, 7 days a week. Today those same budgets are closer to $5,000 per month and some even much lower. Yet the quality of service that the customer of these businesses is expecting continues to climb.

With these demands you need a scaleable sales force, immediate, accurate and secure information exchange, automatic delivery of products, and accurate tracking information for package delivery. In this article we will discuss the issues in constructing a web site that can give you all of this and much more. However, there are some pitfalls to be watchful of. The anonymity of the people buying from you can make you feel like you are talking to Mr. X. You rarely have the chance to speak directly with your customers. It is also far more difficult to get a feel for the size and condition of your vendors and your competitors. You have to help your customers overcome the fear that many people have putting their credit card number into a form on a web page. Using and understanding eCommerce can give you a strong advantage over your competitors while providing greater value and comfort to your customers.

How eCommerce is Different from Normal Business

eCommerce is very similar to the mail-order business. You normally do not have your customer right in front of you to confirm the signatures with the back of their credit card. You do have an advantage in eCommerce in that you can track exactly where your customer is "calling" from. This can help to reduce fraud, unlike mail-order and other types of communication. It also doesn't cost you any more to be open 24 hours a day, 7 days a week.

You have the opportunity to give your customer more information about their purchase, both in terms of product information and in delivery tracking, you can provide direct links to shipping services with tracking numbers and much more. With many of the emerging payment technologies you will also be able to offer smaller priced items for sale, that is, access to a complete new story for 50 cents is now practical in eCommerce, but it would be silly to do with mail-order. If you sell electronic products, like software, your customer doesn't have to wait for overnight delivery services, you can give them immediate delivery once the payment has been cleared.

To make sure that all of this access to information is accurate and secure, there are some precautions that you should take. Don't be like most people and assume that eCommerce is just an electronic catalog on a SSL server. These are the same people that are happy to accept the price for a product from a static HTML page. That would be like honoring a faxed order form that a customer has written their own prices written on; imagine buying a new PowerBook 1400 cs for, oh, how about $10. We will come back to this issue later, first what do we mean by SSL server?

Jay Van Vark is the founder and CEO of Pacific Coast Software a Internet commerce tool and commerce site hosting company, developers of WebCatalog & WebMerchant, marketed by StarNine. Jay has an engineering background and continues to do much of the engineering on the products and services that Pacific Coast Software offers. He is also an active speaker with the MacCryptography conference as well as other Internet & Macintosh conferences. You can reach him at <jayv2@pacific-coast.com>.

What is SSL?

SSL stands for Secure Sockets Layer. This is the technique in which web servers and web browsers encrypt and decrypt all of the information that they transmit and recieve. Secret decoder ring time. Both ends establish and use the same scheme for making sure that no one else is listening to their conversation. Web browsers will typically indicate a secure connection with an alert when the connection is first established and with a key graphic somewhere in the window. As of this writing (August 1997), the only current SSL server implementation available on the Macintosh is from StarNine, WebStar SSL.

SSL encrypts every bit of data that is transmitted from the server to the customer and vice versa. Think about that one, every bit of data, text, pictures and all. This can be very wasteful if you don't use it carefully. Not to mention the fact that there are still some browsers out there that aren't capable of SSL and those users wouldn't be able to access the secure part of your site. You don't want to slam the door on any customers. So now that we have a technique to keep our conversation private, what does the conversation look like?

Flow of a eCommerce transaction

We often describe the web as being analogous to doing business with faxes. Imagine that the home page of your site is a fax back form. You have checkboxes for people to indicate what they are interested in and, in return, you send them another fax. The customer then fills out some more check boxes and we continue the exchange until they get what they want. The web is very similar, the site must respond with a page which elicits more information from the customer guiding them to their buying decision.

The basic flow of a eCommerce site has 4 major sections 1) Entry & Search, 2) Results, 3) Invoice, 4) Thank You. Each of these sections can be a single page on simple sites, or become complete sections on more complicated sites. On the Entry & Search page you must have some way for the customer to select what they want to see. (This actually can be embedded right in the Entry page if you only have a few products, but to keep the flow simple we'll assume that it is its own page.) On the Results page you display information about a product or products and the option to add it to the customer's shopping cart or Invoice. Once you are on the Invoice page you collect the necessary payment information from the customer and complete the order. These steps are illustrated in **Figure 1**.

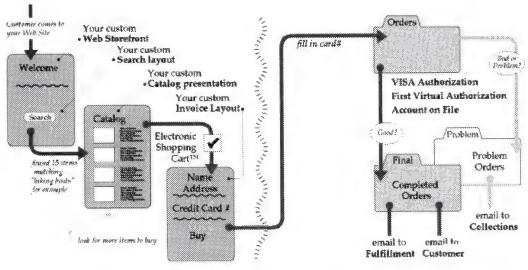


Figure 1. *Flow of a eCommerce Web Site, courtesy of Pacific Coast Software.*

Here are some sites that follow the above flow, just to mention a few:

- Crazy Shirts at <http://www.crazyshirts.com/>.
- Educorp Direct at <http://www.educorp.com/>.
- Club Mac at <http://www.club-mac.com/>.
- CD Direct at <http://www.cddirect.co.uk/>.
- Pro Sound and Lighting at <http://www.cddirect.co.uk/>.
- APCnet CyerbMall at <http://www.apcn.com/>.

As you can tell from this type of flow, all of the pages past the entry page are returned from a CGI. There are a number of commercially available CGIs designed for both database access and the complete eCommerce process, including WebCatalog from StarNine, Tango Merchant from Everyware, Icat from Icat, and many others, both commercial and shareware. You can even write all of this interaction in your own CGI with AppleScript, Frontier or another programming language. Many of the commercial products have their own language to help you. Let's spend just a few minutes talking about the key functions reguired for the CGI to handle the eCommerce transaction.

Tracking the customer

Of primary importance in any transaction is that the customer feel comfortable with your communication. To make it seem like the website is talking to each customer individually you must track who the customer is and what they are interested in. The most common way this is achieved on the web is with the shopping cart concept. This allows many different people to be shopping on your site and all have their own sets of items in their cart. In our fax back example you would have to use something like the fax number to keep track of each customer. The equivalent with the web would be the IP number (known as IP tracking). The one major difference is that a customer's fax number doesn't change very often, while a customer's IP number can change everytime that they connect to the Internet — for those people using dial up accounts or other dynamic addressing situations — so IP numbers are not a very reliable way to track customers.

Another common tracking technique is cookies. You can have your website put a cookie onto the customer's machine so that it maintains important information, like the contents of their shopping cart. A better technique that I have found is tag propagation. This is a technique in which the first page that someone hits when they enter the site assigns a unique number, something like the number of seconds since 1904. This number is in turn passed thru every page on the site and the shopping cart information is stored in a file with that number on the server. This allows a customer to disconnect (by choice or happenstance) from the Internet and not loose the shopping cart information. This can be very important in situations where buying approval from someone else is required for the purchase. Most of the commercial products include a way of doing this. With WebCatalog you insert a cart=[cart] parameter into every HREF and form on your site.

Tracking the customer is very useful not just for the convenience of a shopping cart, but for things like tracking down people that you think are using stolen cards and, more importantly for that all allusive goal, to make the site more usable. Correlating this tracking information with the general web server logs can be used to determine trends of the people visiting your site, are they getting all the information they need to make a buying decision, are they understanding the buying process, are they loosing interest after a certain amount of time. One big advantage of this tracking log is to look for all the searches that people are doing on your site and were they are not finding any products. Maybe you should describe the products more effectively. All of these answers can help you understand ways to change your site to make it more useful.

Calculating Accurate Invoices

The hardest part about calculating invoices is just like fax transactions, you have to wait until you get all of the information from the customer before you can have accurate results. The most obvious place this happens is on the invoice page. Let's just say we had SKU, TITLE, QUANTITY and PRICE on our invoice. In the simplest case the customer gets to the invoice from the result of a search, usually with a simple hyperlink, so you assume a quantity of one. Since you want to allow the customer to order more than one of a product, you make the QUANTITY an input field on the invoice. To provide as much feedback as possible there may be a subtotal and other information on the invoice, so if the user changes the quantity, then they may no longer have accurate information.

This should be the first area of concern for the WebMaster. Once the customer has chosen some products, is the subtotal always accurate? Do they understand what they are looking at? Are there any ways to get the system to accept a "bad" subtotal? If the eCommerce product does not confirm the field "PRICE" from the web page with the value in the database, it will accept whatever the incoming page said the price was. What does that mean? You may have a sneaky customer looking at an invoice for a PowerBook 3400. The original page from the web server says the price is $3799. The sneaky customer can save the web page locally as source from his web browser, open the file with SimpleText and change the price to $10. Now the sneaky customer uses his web browser to view that file, fills in the rest of the purchase information and submits the form to the web server. Obviously, you must make sure that your site uses the "real" price for the PowerBook and not the $10 price!

This is just a simple example of calculation issues; add in taxes and shipping costs and you can see that this can easily get very complex. The best way to overcome these issues is to split the invoice into two pages, a proforma invoice and a final invoice. Most of the commercial products do a very good job taking care of these situations. The proforma invoice shows a listing of the shopping cart, possibly with a subtotal, as well as any other information that you need to complete a final invoice, like quantity for each item, what state they are buying from to help with the tax calculation, choice of shipping method etc. Collecting all of that information will allow you to calculate and display a final invoice. With Lasso and Tango you can communicate back to your current database, SQL, FileMaker etc., to calculate these numbers for your website. Not until you get to the Invoice page is any information sensitive. From this point on, you want to make sure that you are communicating only with the customer. You should make sure that no one is listening in.

Security Concerns

Areas that we DO care about security

As mentioned in the section about SSL we do want to protect the transmission of sensitive information with something like SSL to keep the eavesdroppers away, but another equally important issue for security is protection from attacks on your web server. People trying to find credit card numbers in accounting logs or just trying to steal products, to buy at ridiculously low or free prices. Prevention of this type of security breach is the most overlooked area. Much of the information on the machine should not be allowed any access. You don't want people knowing even about access statistics without you knowing about it.

The first obvious area to secure is the accounting files. Let's say the web server is doing a great job of keeping people out of sensitive areas, but the same machine is also your ftp server. People are prevented by the web server from getting to your accounting log, but maybe there is a security hole because your ftp server software allows access to this log... so my first advice, limit the access protocols to all sensitive data — 1) store your accounting logs and other sensitive files outside of the web server folder, WebStar and many other web server products will not serve files outside of their folder tree, 2) don't run ftp and other protocol services on the same machine. Also, make sure that if you are delivering electronic product, only the person that bought it, gets it. For this you should either be copying the product to some unique place only that person is given access to or have a one time password scheme allowing only one shot at downloading the product.

The concern of the web server allowing access to files that are sensitive is best taken care of by your disk organization. Below is a screen shot of a sample organization of your web server folder structure using WebStar and WebCatalog:

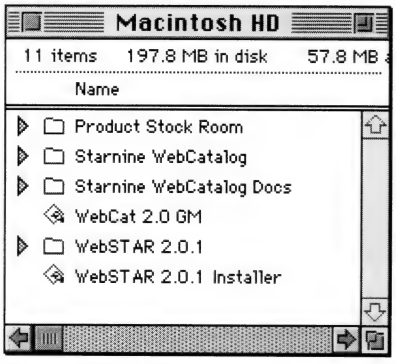


Figure 2. *Folder Structure for a typical web server.*

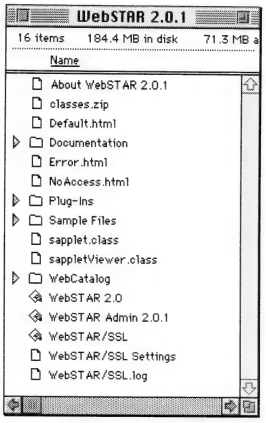


Figure 3. *Folder structure inside WebStar.*

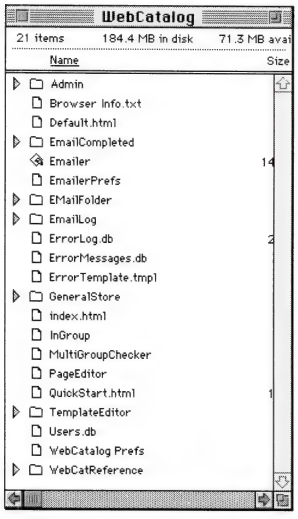


Figure 4. *Folder structure inside WebCatalog.*

Areas that we DON'T care about security

There are many areas within the selection and buying process that are considered public information and therefore don't need security. In fact, the whole process would be slowed down if it sent everything through a SSL server. Imagine if you received a mail-order catalog from MacWarehouse or Club-Mac and you had to put a decoder ring over each letter to figure out what it really was, that would take you hours just to read one page. That is what your browser is doing with SSL data. So, big picture, you only want to use SSL when you are expecting sensitive data from the customer, like a credit card number. Protect that from eavesdroppers with SSL, everything else should go thru the non-SSL server.

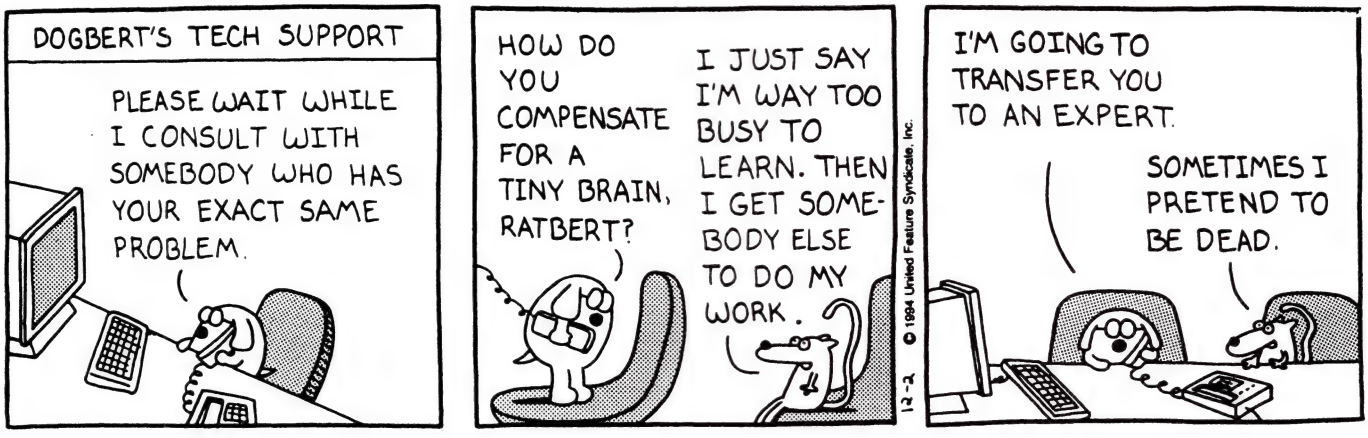
Conclusion

eCommerce is more secure than most business we conduct everyday and is getting better every minute. Knowing various hacking techniques on the Internet and having built an eCommerce package, if I wanted to get a few credit card numbers I would head for the local bar and go thru the dumpster long before I would start going after websites. Give yourself time to understand and work with your new sales force. A properly constructed website benefits the consumer with up to the minute information and immediate response. The same website serves as hundreds of sales people for the merchant, all trained with exactly the right information as well as access to tracking information etc. The positive return for the customer and the merchant will help to overcome the myth and fear of the security on the Internet. I would like to end on a observation about most credit cards, even if it is stolen, the owner is only liable for $50.

There are a variety of tools on the market to help you construct your eCommerce web site. Each has its own strengths and weaknesses. To choose the best for your needs, you must carefully research the speed and responsiveness of the server under load, how they handle the security areas and your database connectivity needs, do they have to handle a live existing database. You can find more information to help you with your research at these web addresses:

- Pacific Coast Software at <http://www.pacific-coast.com>.
- Everyware Inc. at <http://www.everyware.com>.
- BlueWorld Communications at <http://www.blueworld.com>.
- Icat at <http://www.icat.com>.
- StarNine Technologies at <http://www.starnine.com>. MT

Dilbert® *by Scott Adams*



Reprinted by permission on UFS, Inc.

©1997 United Feature Syndicate, Inc. (NYC)

PROGRAMMER'S CHALLENGE

by Bob Boonstra, Westford, MA

Pente®

Reaching once again into the closet where we store board games, I found the game of Pente®, The board physically resembles the board used in GO, but the game strategies are simpler. Pente® is played by two players who alternate placing stones on a 19x19 grid. The objective is to win the game by getting five or more stones in a row or, alternatively, by capturing five or more pairs of your opponent's stones. Your Challenge is to write code that will play the game of Pente® and accumulate the most points (described below) in the minimum time.

The prototype for the code you should write is:

```
typedef struct Capture {
  Point stone1;
  Point stone2;
} Capture;

void InitPente(
  long boardHalfSize          /* e.g., 9 for a 19x19 board */
                              /* all coordinates between -boardHalfSize
                              and +boardHalfSize */
);

void Pente(
  Point opponentsMove,        /* your opponent moved here */
  Boolean playingFirst,       /* ignore opponentMove */
  Point *yourMove,            /* return your move here */
  Capture claimCaptures[],    /* return coordinates of captured pairs here */
  long *numCaptures,          /* return number of claimCaptures here */
  Boolean *claimVictory       /* return true if you claim victory with this move */
);

void TermPente(void);         /* deallocate any dynamic storage */
```

Captures take place by bracketing two adjacent stones of your opponents. Given the position

```
---BWW---
```

... Black can capture the two White stones by playing ...

```
---BWWB--
```

... after which the two White stones are removed ...

```
---B  B--
```

Captures can occur horizontally, vertically, or diagonally. Not that no capture occurs if White moves into the unoccupie square below:

```
---BW-B--
```

Multiple captures can occur on a single move.

The game ends when one side captures five pairs of th opponent's stones, or when one side places five stones in straight line, either horizontally, vertically, or diagonally. Whe one side obtains an unblocked row of four stones, called tessera, a win is imminent. Therefore, an unblocked row of thre stones, called a tria, is a serious threat that should be blocke unless a stronger offensive move exists.

To neutralize the advantage that the first player has, the firs player's second move is restricted to be three or mor intersections away from the center of the board (i.e., the `h` an `v` coordinates of White's second move must both be greater tha or equal to 3 in absolute value).

At the start of the game, your **InitPente** routine (and that o your opponent) will be called with the half-width of the board in **boardHalfSize** (between 9 and 15, inclusive). The **Pente** routine will then be alternately called, providing the location of you **opponentsMove** (unless you are **playingFirst**). You should place a stone in an unoccupied location and return that location in **yourMove**. In addition, if your move captures any adjacent pairs of your opponent's stones, you should report the number of captures in **numCaptures**, and the locations of the captured pairs in **claimCaptures**. If your move results in victory, either by achieving 5 of your stones in a row, or a cumulative capture of 5 pairs of your opponent's stones, you should set **claimVictory** to **true**. At the end of the game, `TermPente` will be called, where you should deallocate any dynamically allocated storage.

THE RULES

Here's how it works: each month we present a new programming challenge. First, write some code that solves the challenge. Second, optimize your code (a lot). Then, submit your solution to MacTech Magazine. We choose a winner based on code correctness, speed, size, and elegance (in that order of importance) as well as the submission date. In the event of multiple equally desirable solutions, we'll choose one winner (with honorable mention, but no prize, given to the runner up). The prize for each month's best solution is a $100 credit for Developer Depot™.

Unless stated otherwise in the problem statement, the following rules apply: All solutions must be in ANSI compatible C or C++, or in Pascal. We disqualify entries with any assembly in them (except for challenges specifically stating otherwise.) You may call any Macintosh Toolbox routine (e.g., it doesn't matter if you use NewPtr instead of malloc). We compile all entries into native PowerPC code with compiler options set to enable all available speed optimizations. The development environment to be used for selecting the winner will be stated in the problem. **Limit your code to 60 characters per line** or compress and binhex the solution; this helps with e-mail gateways and page layout.

We publish the solution and winners for each month's Programmer's Challenge three months later. All submissions must be received by the 1st day of the month printed on the front cover of this issue.

You can get a head start on the Challenge by reading the Programmer's Challenge mailing list. It will be posted to the list on or before the 12th of the preceding month. To join, send an email to listserv@listmail.xplain.com with the subject "subscribe challenge-A".

Mark solutions "Attn: Programmer's Challenge Solution" and send it by e-mail to one of the Programmer's Challenge addresses in the "How to Communicate With Us" section on page 2 of this issue. Include the solution, all related files, and your contact info.

MacTech Magazine reserves the right to publish any solution entered in the Programmer's Challenge. Authors grant MacTech Magazine the exclusive right to publish entries without limitation upon submission of each entry. Authors retain copyrights for the code.

Board coordinates are expressed as distance from the center square in ordered (v,h) pairs, so that the center intersection is at (0,0), and the corners of the standard board are at (-9,-9), (-9,9), etc.

At the end of the game, points will be awarded as follows:

5 points for the player who achieved 5 stones in a row
1 point for each capture
1 point for each distinct sequence of 4 stones in a row remaining on the board

One point will be deducted for each second of execution time used during a player's turns, including the time taken by the InitPente and TermPente routines. Both the game winner and the game loser will accumulate points. It is possible to earn negative points. The Challenge winner will be the entry that accumulates the most points in a round-robin tournament of all entries, where each entry plays each other entry at least twice, half of the time playing first and half playing second.

Your code may allocate up to 10MB of dynamic storage, including both explicit calls to NewPtr/malloc and allocation of dynamic objects. This will be a native PowerPC Challenge, using the latest CodeWarrior environment. Solutions may be coded in C, C++, or Pascal.

Pente® is published by Pente Games, Inc.

Three Months Ago Winner

Congratulations to Peter Lewis (Perth, Australia) for submitting the winning entry to the Stratego Challenge. Peter's entry was a convincing winner in a tournament of the 5 entries submitted.

The Stratego tournament consisted of 80 games, with each entry playing against each other entry 8 times, 4 times playing first and 4 times playing second. Peter's entry plays a very good game of Stratego, as evidenced by the fact that it won 30 of the 32 games it played. His algorithm, described in the commentary at the beginning of his code, includes aggressively attempting to capture weaker pieces, exploring with low ranking pieces of his own, running away from stronger pieces, exchanging pieces of equal rank, and advancing toward unknown pieces. Of the 30 games Peter's entry won, 10 were won by capturing the opponent's flag, and 19 were won by eliminating all of the pieces that the opponent could move. (The remaining win resulted from resignation of the opponent.)

Several of the entries used delaying tactics in an attempt to take advantage of the scoring rules and force their opponent to earn negative points for a win. As discussed on the Challenge mailing list, I cut off games where one player took longer than 20 seconds, declared a tie, and awarded points (sometimes negative points) to each player.

The table below lists the tournament results and point totals for each entry. The number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges to date prior to this one.

Name	Wins	Points	Code	Data	Language
Peter Lewis (37)	30	296.62	13436	310	C++
Dennis Jones	13	86.75	8632	414	C
Randy Boring (39)	5	58.68	5724	618	C
Ernst Munter (286)	9	29.45	12044	4072	C++
Tom Saxton (10)	7	-51.69	7320	460	C

Top 20 Contestants

Here are the Top Contestants for the Programmer's Challenge. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	200	11.	Antoniewicz, Andy	24
2.	Gregg, Xan	63	12.	Picao, Miguel Cruz	21
3.	Lewis, Peter	57	13.	Day, Mark	20
4.	Cooper, Greg	54	14.	Higgins, Charles	20
5.	Boring, Randy	41	15.	Studer, Thomas	20
6.	Lengyel, Eric	40	16.	Saxton, Tom	17
7.	Mallett, Jeff	30	17.	Gundrum, Eric	15
8.	Murphy, ACC	30	18.	Hart, Alan	14
9.	Nicolle, Ludovic	28	19.	O'Connor, Turlough	14
10.	Larsson, Gustav	27	20.	Karsh, Bill	12

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place .....20 points
2nd place .....10 points
3rd place .....7 points
4th place.....4 points
5th place .....2 points
finding bug .....2 points
suggesting Challenge...2 points

Here is Peter's winning solution:

Challenge.cp

© 1997 Peter N Lewis

```
#define ASSERTIONS 0
#define DEBUG_RULES 0

#include <Timer.h>
#include <stdlib.h>
#include <string.h>

#include "Challenge.h"

/*
    Author: Peter N Lewis

    Assumptions:
        Only time we spend thinking is counted against out 10 seconds (not time in
GetMove/ReportMove)
        [Actually, this assumption is not valid, but Peter won anyway. —Bob]

    Method:
        Basically we keep track of the board and what we know and what they know.
        Each opponent piece has a bit map associated with it describing what pieces it
could be. As we see more pieces, the bit map is culled. If the piece moves, the bomb
& flag bits are removed. If we've seen all Scouts (for example), then the Scout bit is
```

removed from all remaining pieces. If all but one bit is remvoed, then we know what the piece is.

At each turn, we simply apply a sequence of actions (listed below) and take the first action that works.

It does very little in the way of lookahead (it plans out a path, but doesn't remember it and doesn't take it to account any movement by the opposition)

It keeps a CRC of recent board positions (since the last strike) and doesn't replay any boards (we want to win, not draw!).

If we exceed 10 seconds thinking time, we resign. Not that this is particularly likely, in the games I tried, it spend less than half a second total.

Optimizations:
None.

Comment:
It actually plays a half decent game! The end game is not as good as I'd like, but time is up!
*/

/*
USE SPY
If our spy is next to their 1, kill it

DEFEND AGAINST SPY
if we have seen the spy, ignore this case

If an unknown piece is next to the 1, then run, attack, have another piece attack, or ignore depending on a table

ATTACK WEAKER
If a known piece is next to a weaker known piece, attack it except if it places that piece in a dangerous location

EXPLORE ATTACK
If a 6,7,9 is next to an unknown piece, attack it

RETREAT
If a known piece is next to a stronger known piece, run away (preferably towards something that can kill it or if it's lowly, towards an unknown piece)

SCOUT
Try advancing scouts rapidly

ATTACK DISTANT
If a known piece is distant, but a clear path leads a slightly better piece towards it, advance the better piece (includes miners)

EXPLORE DISTANT
Try exploring (advance lowly pieces towards unknown pieces)

ATTACK KNOWN WITH SAME DISTANT
If a known piece can be attacked by a known identical piece, attack it

FIND FLAG
When few unmoved pieces remain, start assuming they are bombs/flags

MOVE FORWARD
Move any piece we can forward

MOVE
Move any piece we can

RESIGN
Give up
*/

```
#if ASSERTIONS

static void Assert( short must )
{
  if ( !must ) {
    DebugStr( "\pAssert failed!\n" );
  }
}

#else

#define Assert( must )

#endif

enum {
  kEmpty = kFlag+1,
  kWater,
  kMoved,          // fake rank for moved pieces
  kAddForRankish   // add this in for enemies when calculating the CRC
};

enum {
  kNoColor = 0
};

enum {
  kNoNothing = 0x00001FFE,
  kStationaryBits = ((1 << kBomb) | (1 << kFlag))
};

enum {
  kRepeatedBoards = 1000
};

typedef struct Square {
  PlayerColor color;
  PieceRank rank;
  UInt32 possibilities;
} Square;

typedef Square OurBoard[kBoardSize][kBoardSize];

typedef int Counts[kFlag+1];

typedef UInt32 BoardPossibilities[kBoardSize][kBoardSize];

typedef struct Storage {
  UInt32 total_time;
  UInt32 extra_time;
  OurBoard board;
  Counts our_pieces;
  Counts their_pieces;
  Boolean do_getmove;
  Boolean victory;
  Square blankSquare;
  PlayerColor playerColor;
  PlayerColor theirColor;
  BoardPossibilities dangers;
  BoardPossibilities known_dangers;
  UInt32 repeated_board[kRepeatedBoards];
  UInt32 repeated_board_count;
} Storage, *StoragePtr;

static char *board_setup[4] = {
// 1 = Marshal, ..., 9 = Scout, : = Spy, ; = Bomb, < = Flag
  "8;<;67;7;7",
  "48;3862;89",
  "6359954865",
  "997159:499",
};

static char *start_piece_counts = "0112344458161";

static int dR[4] = { 1, 0, -1, 0 };
static int dC[4] = { 0, -1, 0, 1 };

#if ASSERTIONS
```

AssertValidBoard

```
static void AssertValidBoard( StoragePtr storage )
{
  int piece;
  int count1 = 0;
  int count2 = 0;
  int row, col;

  for ( piece = kMarshall; piece <= kFlag; piece++ ) {
    count1 += storage->their_pieces[piece];
  }

  for ( row = 0; row < kBoardSize; row++ ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( storage->board[row][col].color == storage-
>theirColor
          && storage->board[row][col].rank == kUnknown ) {
        count2++;
      }
    }
  }
```

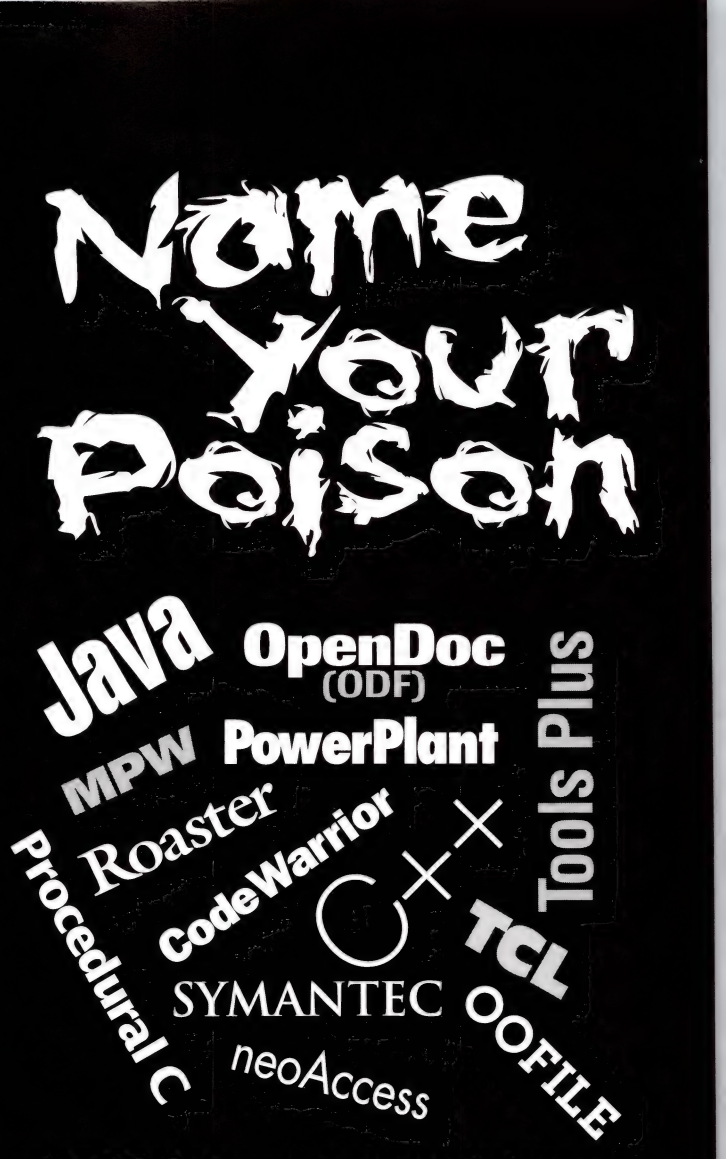
```
 Assert( count1 == count2 );

else

define AssertValidBoard( storage )

endif
```

PositionPieces

```
oid PositionPieces(
 void *privStorage,          /* 1MB of preinitialized storage for your use */
 PlayerColor playerColor,    /* you play red or blue, with red playing first */
 Board *theBoard             /* provide the initial position of your pieces */


 StoragePtr storage = (StoragePtr) privStorage;
 int row, our_row, their_row, col, board_col;
 PlayerColor theirColor;
 int piece;
 Boolean reverse = (TickCount() & 1) != 0;

 Assert( strlen(board_setup[0]) == kBoardSize );
 Assert( strlen(board_setup[1]) == kBoardSize );
 Assert( strlen(board_setup[2]) == kBoardSize );
 Assert( strlen(board_setup[3]) == kBoardSize );

 for ( row = 0; row <= 3; row++ ) {
   if ( playerColor == kRed ) {
     our_row = row;
     their_row = (kBoardSize-1)-row;
     theirColor = kBlue;
   } else {
     their_row = row;
     our_row = (kBoardSize-1)-row;
     theirColor = kRed;
   }
   for ( col = 0; col < 10; col++ ) {
     board_col = reverse ? (kBoardSize-1) - col : col;
     (*theBoard)[our_row][col].thePieceRank = (PieceRank)
           (board_setup[row][board_col] - '0');
     (*theBoard)[our_row][col].thePieceColor = playerColor;

     storage->board[our_row][col].color = playerColor;
     storage->board[our_row][col].rank =
           (*theBoard)[our_row][col].thePieceRank;
     storage->board[our_row][col].possibilities = kNoNothing;

     storage->board[their_row][col].color = theirColor;
     storage->board[their_row][col].rank = kUnknown;
     storage->board[their_row][col].possibilities = kNoNothing;
   }
 }

 for ( row = 4; row <= 5; row++ ) {
   for( col = 0; col < kBoardSize; col++ ) {
     storage->board[row][col].color = (PlayerColor)kNoColor;
     storage->board[row][col].rank =
          (PieceRank) ((col/2 % 2 == 1) ? kWater : kEmpty);
     storage->board[row][col].possibilities = 0;
   }
 }

 for ( piece = kMarshall; piece <= kFlag; piece++ ) {
   storage->our_pieces[piece] =
         start_piece_counts[piece] - '0';
   storage->their_pieces[piece] =
         start_piece_counts[piece] - '0';
 }

 storage->do_getmove = (playerColor == kBlue);
 storage->victory = false;
 storage->blankSquare = storage->board[4][0];
 storage->playerColor = playerColor;
 storage->theirColor = playerColor == kRed ? kBlue : kRed;
 storage->repeated_board_count = 0;

 AssertValidBoard( storage );
}
```

Learn

```
static void Learn( StoragePtr storage, Boolean them,
                          int row, int col, PieceRank rank )
```

AppMaker supports most popular languages and frameworks. Just point and click to design your user interface, then let AppMaker create resources and generate "human, professional quality code" to implement your design.

Use AppMaker as a prototyping and productivity tool or use it as a learning tool for Mac programming techniques or for new environments such as OpenDoc, Java, or PowerPlant.

AppMaker is just $299 and includes a one-year subscription on CD.

B•O•W•E•R•S Development

603-863-0945
bowersdev@aol.com
http://members.aol.com/bowersdev

```
{
  Boolean gotall;
  PlayerColor thiscolor;
  int r, c;

  if ( storage->board[row][col].rank == kUnknown ) {

    if ( rank == kMoved ) {
      UInt32 possibilities =
            storage->board[row][col].possibilities;
      possibilities &= ~kStationaryBits;

      if ( (possibilities & (possibilities-1)) == 0 ) {
              // only one bit on! Now we know!
        int newrank;
        newrank = 0;
        while ( (possibilities & 1) == 0 ) {
          possibilities >>= 1;
          newrank++;
        }
        rank = (PieceRank)newrank;
      } else {
        storage->board[row][col].possibilities = possibilities;
      }
    }

    if ( rank != kMoved ) {
      storage->board[row][col].rank = rank;
      storage->board[row][col].possibilities = (1 << rank);
      if ( them ) {
        gotall = -storage->their_pieces[rank] == 0;
      } else {
        gotall = -storage->our_pieces[rank] == 0;
      }
      if ( gotall ) {
        thiscolor = storage->board[row][col].color;
        for ( r = 0; r < kBoardSize; r++ ) {
          for ( c = 0; c < kBoardSize; c++ ) {
            if ( storage->board[r][c].rank == kUnknown
                 && storage->board[r][c].color == thiscolor )
{
              UInt32 possibilities =
                 storage->board[r][c].possibilities;
            possibilities &= ~ (1 << rank);
            storage->board[r][c].possibilities = possibilities;
              if ( (possibilities & (possibilities-1)) == 0 )
{
                      // only one bit on!
                int newrank;
                newrank = 0;
                while ( (possibilities & 1) == 0 ) {
                  possibilities >>= 1;
                  newrank++;
                }
            Learn( storage, them, r, c, (PieceRank)newrank );
                }
              }
            }
          }
        }
      }
    }
  } else {
    Assert( rank == kMoved ||
              storage->board[row][col].rank == rank );
  }
}
```

HandleTheirMove

```
static void HandleTheirMove( StoragePtr storage,
                     GetOpponentMove GetMove )
{
  PiecePosition moveFrom;
  PiecePosition moveTo;
  Boolean moveStrike;
  MoveResult moveResult;
  UnsignedWide start, finish;

  Microseconds( &start );
  (*GetMove)( &moveFrom,&moveTo, &moveStrike,&moveResult );
  Microseconds( &finish );
  storage->extra_time += finish.lo - start.lo;

  Assert( moveResult.legalMove );
    // They must have made a legal move or we would not be called
  Assert( !moveResult.victory );
    // If they won we would not be called
  if ( moveStrike ) {
    Learn( storage, true, moveFrom.row, moveFrom.col,
           moveResult.rankOfAttacker.thePieceRank );
    Learn( storage, false, moveTo.row, moveTo.col,
           moveResult.rankOfDefender.thePieceRank );
    if ( moveResult.attackerRemoved &&
             moveResult.defenderRemoved ) {
      storage->board[moveFrom.row][moveFrom.col] =
           storage->blankSquare;
      storage->board[moveTo.row][moveTo.col] =
           storage->blankSquare;
    } else if ( moveResult.attackerRemoved ) {
    if (storage->board[moveTo.row][moveTo.col].rank == kBomb)
{
        storage->board[moveFrom.row][moveFrom.col] =
          storage->blankSquare;
      } else {
        storage->board[moveFrom.row][moveFrom.col] =
            storage->board[moveTo.row][moveTo.col];
        storage->board[moveTo.row][moveTo.col] =
            storage->blankSquare;
      }
    } else {
      Assert( moveResult.defenderRemoved );
      storage->board[moveTo.row][moveTo.col] =
          storage->board[moveFrom.row][moveFrom.col];
      storage->board[moveFrom.row][moveFrom.col] =
          storage->blankSquare;
    }
  } else {
    storage->board[moveTo.row][moveTo.col] =
        storage->board[moveFrom.row][moveFrom.col];
    storage->board[moveFrom.row][moveFrom.col] =
        storage->blankSquare;
    if ( abs(moveTo.row - moveFrom.row) +
             abs(moveTo.col - moveFrom.col) > 1 ) {
      Learn( storage, true, moveTo.row, moveTo.col, kScout );
    } else {
      Learn( storage, true, moveTo.row, moveTo.col,
          (PieceRank)kMoved );
    }
  }

  AssertValidBoard( storage );
}
```

FindPiece

```
static Boolean FindPiece( StoragePtr storage, PlayerColor
color,
                                        PieceRank rank, int
*row, int *col )
{
  int r, c;

  for ( r = 0; r < kBoardSize; r++ ) {
    for( c = 0; c < kBoardSize; c++ ) {
      if ( storage->board[r][c].color == color
           && storage->board[r][c].rank == rank ) {
        *row = r;
        *col = c;
        return true;
      }
    }
  }
  return false;
}
```

IsOnBoardWeak

```
static Boolean IsOnBoardWeak( int row, int col )
{
  return 0 <= row && row < kBoardSize &&
            0 <= col && col < kBoardSize;
}
```

IsOnBoard

```
tatic Boolean IsOnBoard( int row, int col )

 if (   0 <= row && row < kBoardSize &&
        0 <= col && col < kBoardSize ) {
   if ( row <= 3 || row >= 6 ) {
     return true;
   }
   if ( col <= 1 || col >= 8 ) {
     return true;
   }
   if ( 4 <= col && col <= 5 ) {
     return true;
   }
 }
 return false;
```

IsColorPiece

```
tatic Boolean IsColorPiece( StoragePtr storage,
                                    int row, int col,
layerColor color )

  Assert( IsOnBoardWeak( row, col ) );
  return storage->board[row][col].color == color;
```

IsOurPiece

```
static Boolean IsOurPiece( StoragePtr storage, int row, int
col )
{
  Assert( IsOnBoardWeak( row, col ) );
  return storage->board[row][col].color == storage-
>playerColor;
}
```

IsTheirPiece

```
static Boolean IsTheirPiece( StoragePtr storage,
                                    int row, int col )
{
  Assert( IsOnBoardWeak( row, col ) );
  return storage->board[row][col].color == storage-
>theirColor;
}
```

IsUnknownPiece

```
static Boolean IsUnknownPiece( StoragePtr storage,
                                    int row, int col )
{
  Assert( IsOnBoardWeak( row, col ) );
  return storage->board[row][col].rank == kUnknown;
}
```

IsRankPiece

```
static Boolean IsRankPiece( StoragePtr storage,
                                    int row, int col,
PieceRank rank )
{
  Assert( IsOnBoardWeak( row, col ) );
  return storage->board[row][col].rank == rank;
}
```

IsEmptySquare

```
static Boolean IsEmptySquare( StoragePtr storage,
                                    int row, int col )
{
  Assert( IsOnBoardWeak( row, col ) );
  return storage->board[row][col].rank == (PieceRank)kEmpty;
}
```

IsWaterSquare

```
static Boolean IsWaterSquare( StoragePtr storage,
                                    int row, int col )
{
  Assert( IsOnBoardWeak( row, col ) );
  return storage->board[row][col].rank == (PieceRank)kWater;
}
```

IsLowlyRank

```
static Boolean IsLowlyRank( PieceRank rank )
{
  return kCaptain <= rank && rank <= kScout && rank != kMiner;
}
```

IsLowlyPiece

```
static Boolean IsLowlyPiece( StoragePtr storage,
                                    int row, int col )
{
  Assert( IsOnBoard( row, col ) );
  return IsLowlyRank( storage->board[row][col].rank );
}
```

IsMovedPiece

```
static Boolean IsMovedPiece( StoragePtr storage,
                                    int row, int col )
{
  Assert( IsOnBoard( row, col ) );
  return (storage->board[row][col].possibilities &
        kStationaryBits) == 0;
}
```

IsRevealedPiece

```
static Boolean IsRevealedPiece( StoragePtr storage,
                                    int row, int col )
{
  Assert( IsOnBoard( row, col ) );
  Assert( IsOurPiece( storage, row, col ) );
  UInt32 possibilities = storage-
>board[row][col].possibilities;
  return ( (possibilities & (possibilities-1)) == 0 );
}
```

CountAdjacentUnknownPieces

```
static int CountAdjacentUnknownPieces( StoragePtr storage,
                                    PlayerColor color, int
row, int col )
{
  int d;
  int unknowns = 0;

  for ( d = 0; d < 4; d++ ) {
    int r = row + dR[d];
    int c = col + dC[d];

    if ( IsOnBoard( r, c ) && IsColorPiece( storage, r, c,
      color )
                        && IsUnknownPiece( storage, r, c ) )
{
      unknowns++;
    }
  }

  return unknowns;
}

static char *defend_spy_table =
"RARROAOORARRRARRXAXAOAOOXAXAXAXA";
// Run/Attack/Other/Nothing, >1 unknown:other:danger:moved
```

LowlyCanAttack

```
static Boolean LowlyCanAttack( StoragePtr storage, int row,
                    int col, int *otherRow, int *otherCol )
{
  for ( int d = 0; d < 4; d++ ) {
    int r = row + dR[d];
    int c = col + dC[d];

    if ( IsOnBoard( r, c )
        && IsOurPiece( storage, r, c )
        && IsLowlyPiece( storage, r, c ) ) {
      *otherRow = r;
      *otherCol = c;
      return true;
    }
  }
  return false;
}
```

UpdateDangerPossibilities

```
static void UpdateDangerPossibilities( StoragePtr storage )
{
  int row, col;

  for ( row = 0; row < kBoardSize; row++ ) {
    for( col = 0; col < kBoardSize; col++ ) {
      storage->dangers[row][col] = 0;
      storage->known_dangers[row][col] = 0;
    }
  }
  for ( row = 0; row < kBoardSize; row++ ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsTheirPiece( storage, row, col ) ) {
        UInt32 possibilities =
            (storage->board[row][col].possibilities &
                ~kStationaryBits);
        UInt32 known_possibilities = 0;

        if ( storage->board[row][col].rank != kUnknown ) {
          known_possibilities = possibilities;
        }

        for ( int d = 0; d < 4; d++ ) {
          int r = row + dR[d];
          int c = col + dC[d];

      if ( IsOnBoard( r, c ) ) {
        storage->dangers[r][c] |= possibilities;
        storage->known_dangers[r][c] |= known_possibilities;
          }
        }
      }
    }
  }
}
```

GetDangerPossibilities

```
static UInt32 GetDangerPossibilities( StoragePtr storage,
                            int row, int col )
{
  Assert( IsOnBoard( row, col ) );
  return storage->dangers[row][col];
}
```

PossibilitiesCouldKill

```
static Boolean PossibilitiesCouldKill( PieceRank rank,
                            UInt32 possibilities )
{
  if ( (possibilities & ~kStationaryBits) == 0 ) {
    return false;
  }

  switch ( rank ) {
    case kFlag:
      return true;
    case kBomb:
      return (possibilities & (1 << kMiner)) != 0;
    case kMarshall:
      return (possibilities & ((1 << kMarshall) + (1<< kSpy)))
                                                    != 0;
    default:
      return (possibilities & ((1 << (rank+1)) - 1)) != 0;
  }
}
```

PossibilitiesCouldKillSafely

```
static Boolean PossibilitiesCouldKillSafely( PieceRank rank,
                            UInt32 possibilities )
{
  if ( (possibilities & ~kStationaryBits) == 0 ) {
    return false;
  }

  switch ( rank ) {
    case kFlag:
      return true;
    case kBomb:
      return (possibilities & (1 << kMiner)) != 0;
    case kMarshall:
      return (possibilities & ((1<< kSpy))) != 0;
    default:
      return (possibilities & ((1 << rank) - 1)) != 0;
  }
}
```

WillKillPossibilitie

```
static Boolean WillKillPossibilities( PieceRank rank,
                            UInt32 possibilities )
{
  Assert( possibilities != 0 );

  switch ( rank ) {
    case kFlag:
      return false;
    case kBomb:
      return false;
    case kMiner:
      return (possibilities & ~((1 << kScout) + (1 << kBomb) +
                                    (1 << kFlag))) == 0;
    case kSpy:
      return (possibilities & ~(1 << kMarshall)) == 0;
    default:
      return (possibilities & (((1 << (rank + 1)) - 1) +
                                    (1 << kBomb))) == 0;
  }
}
```

WillKillOrSuicidePossibilities

```
static Boolean WillKillOrSuicidePossibilities( PieceRank rank,
                            UInt32 possibilities )
{
  Assert( possibilities != 0 );

  switch ( rank ) {
    case kFlag:
      return false;
    case kBomb:
      return false;
    case kMiner:
      return (possibilities & ~((1 << kScout) + (1 << kMiner) +
                        (1 << kBomb) + (1 << kFlag))) == 0;
    case kSpy:
    return (possibilities & ~((1 << kMarshall) + (1 << kSpy)))
                                        == 0;
    default:
    return (possibilities & (((1 << rank) - 1) + (1 << kBomb)))
                                                    == 0;
  }
}
```

WillPossibilitiesKill

```
static Boolean WillPossibilitiesKill( UInt32 possibilities,
                            PieceRank rank )
{
  Assert( possibilities != 0 );
  possibilities &= ~kStationaryBits;
  if ( possibilities == 0 ) {
    return false;
  }

  switch ( rank ) {
    case kFlag:
      return true;
    case kBomb:
      return possibilities == (1 << kMiner);
    default:
      return (possibilities & ~((1 << (rank+1))-1)) == 0;
  }
}
```

FindSafeSquare

```
static Boolean FindSafeSquare( StoragePtr storage, int row,
                        int col, int *safeRow, int *safeCol )
{
  Assert( IsOnBoard( row, col ) );

  PieceRank rank = storage->board[row][col].rank;
  int doff = (storage->playerColor == kBlue ? 0 : 2);
        // Try backwards first

  for ( int d = 0; d < 4; d++ ) {
    int dr = dR[(d + doff) % 4];
    int dc = dC[(d + doff) % 4];
```

```
    int r = row + dr;
    int c = col + dc;

    while ( IsOnBoard( r, c ) &&
            IsEmptySquare( storage, r, c ) ) {
      if ( !PossibilitiesCouldKill( rank,
           GetDangerPossibilities( storage, r, c ) ) ) {
        *safeRow = r;
        *safeCol = c;
        return true;
      }
      if ( rank != kScout ) {
        break;
      }
      r += dr;
      c += dc;
    }
  }
  return false;
}
```

CountEnemies

```
static void CountEnemies( StoragePtr storage, int row, int
col,
                          int *knowns, int *unknowns )
{
  *knowns = 0;
  *unknowns = 0;

  for ( int d = 0; d < 4; d++ ) {
    int r = row + dR[d];
    int c = col + dC[d];

    if ( IsOnBoard( r, c ) && IsTheirPiece( storage, r, c )
) {
      if ( storage->board[r][c].rank == kUnknown ) {
        *unknowns += 1;
      } else {
        *knowns += 1;
      }
    }
  }
}

/*
static Boolean CanRun( StoragePtr storage, int row, int col,
int *runRow, int *runCol )
{
  for ( int d = 0; d < 4; d++ ) {
    int r = row +
            dR[(d + (storage->playerColor == kBlue ? 0 :
2)) % 4];
                  // Try backwards first
    int c = col +
            dC[(d + (storage->playerColor == kBlue ? 0 :
2)) % 4];

    if ( IsOnBoard( r, c ) &&
         (storage->board[r][c].rank == kEmpty) ) {
      *runRow = r;
      *runCol = c;
      return true;
    }
  }
  return false;
}
*/
```

FindSafePath

```
static Boolean FindSafePath( StoragePtr storage,
      Boolean very_safe, Boolean suicide_ok, int from_row,
      int from_col, int to_row, int to_col, int *best_path,
      int *first_row, int *first_col )
{
  Assert( IsOurPiece( storage, from_row, from_col ) );

  PieceRank rank = storage->board[from_row][from_col].rank;
  BoardPossibilities *dangers =
    very_safe ? &storage->dangers : &storage->known_dangers;

  if (  abs( from_row - to_row ) +
        abs( from_col - to_col ) > *best_path ) {
    return false;
  }

  if ( abs( from_row - to_row ) +
       abs( from_col - to_col ) == 1 ) {
    *best_path = 0;
    *first_row = to_row;
    *first_col = to_col;
    return true;
  }

  int path_length_to[kBoardSize][kBoardSize];
  PiecePosition que[kBoardSize * kBoardSize];
  int que_start = 0;
  int que_fin = 0;
  int que_next_len = 0;
  int current_len = 0;
  int row, col;

  for ( row = 0; row < kBoardSize; row++ ) {
    for( col = 0; col < kBoardSize; col++ ) {
      path_length_to[row][col] = -1;
    }
  }

  que[que_fin].row = from_row;
  que[que_fin].col = from_col;
  path_length_to[from_row][from_col] = 0;
  que_fin++;
  que_next_len = que_fin;

  while ( que_fin > que_start ) {
    row = que[que_start].row;
    col = que[que_start].col;
    que_start++;

    for ( int d = 0; d < 4; d++ ) {
      int dr = dR[d];
      int dc = dC[d];
// scout moves NYI
      int r = row + dr;
      int c = col + dc;

      if ( IsOnBoard( r, c ) && path_length_to[r][c] == -1
           && IsEmptySquare( storage, r, c ) ) {
        if ( suicide_ok ?
          !PossibilitiesCouldKillSafely( rank,
              (*dangers)[r][c] )
          : !PossibilitiesCouldKill( rank, (*dangers)[r][c]
                ) ) {
          path_length_to[r][c] = current_len + 1;
          if ( abs( to_row - r ) + abs( to_col - c ) == 1 )
{
            *best_path = current_len + 1;
            while ( current_len > 0 ) {
              for ( int d = 0; d < 4; d++ ) {
                int backr = r + dR[d];
                int backc = c + dC[d];

                if ( path_length_to[backr][backc] ==
                     current_len ) {
                  r = backr;
                  c = backc;
                  break;
                }
              }
              current_len--;
            }
            *first_row = r;
            *first_col = c;
            return true;
          }
          que[que_fin].row = r;
          que[que_fin].col = c;
          que_fin++;
        } else {
          path_length_to[r][c] = 1000; // Cant go here
        }
      }
    }

    if ( que_start == que_next_len ) {
      que_next_len = que_fin;
```

```
      current_len++;
    }
  }

  return false;
}
```

CalcBoardCRC

```
static UInt32 CalcBoardCRC( StoragePtr storage,
              int from_row, int from_col, int to_row, int
to_col )
{
  Assert( !IsOnBoard( from_row, from_col ) ||
        IsOurPiece( storage, from_row, from_col ) );
  Assert( !IsOnBoard( to_row, to_col ) ||
        IsEmptySquare( storage, to_row, to_col ) );

  UInt32 result = 0;

  int row, col;
  int rankish;

  for ( row = 0; row < kBoardSize; row++ ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( row == from_row && col == from_col ) {
        rankish = 0;
      } else if ( row == to_row && col == to_col ) {
        rankish = storage->board[from_row][from_col].rank;
      } else if ( IsEmptySquare( storage, row, col ) ||
                  IsWaterSquare( storage, row, col ) )
{
        rankish = 0;
      } else if ( IsOurPiece( storage, row, col ) ) {
        rankish = storage->board[row][col].rank;
      } else {
        rankish = storage->board[row][col].rank +
          kAddForRankish;
      }
      result += rankish; // Hmm, not a very good CRC
      result = result * 11 + (result >> 25);
    }
  }

  return result;
}
```

OKMove

```
static Boolean OKMove( StoragePtr storage,
  int from_row, int from_col, int to_row, int to_col )
{
  if ( IsTheirPiece( storage, to_row, to_col ) ) {
    return true;
  }

  UInt32 crc = CalcBoardCRC( storage, from_row, from_col,
                                        to_row, to_col );
  long i;
  for ( i = 0; i < storage->repeated_board_count; i++ ) {
    if ( crc == storage->repeated_board[i] ) {
      return false;
    }
  }
  return true;
}
```

AppendRepeatedBoard

```
static void AppendRepeatedBoard( StoragePtr storage )
{
  UInt32 crc = CalcBoardCRC( storage, -1, -1, -1, -1 );

  if ( storage->repeated_board_count == kRepeatedBoards ) {
    storage->repeated_board_count--;
    BlockMoveData( &storage->repeated_board[1],
                   &storage->repeated_board[0],
                   storage->repeated_board_count *
                 sizeof(storage->repeated_board[0]) );
  }
  storage->repeated_board[storage->repeated_board_count++] = crc;
}

#if DEBUG_RULES
  #define RETURN( x ) DebugStr( x ";g" ); return
#else
  #define RETURN( x ) return
#endif
```

FigureOutOurMove

```
static void FigureOutOurMove( StoragePtr storage,
                PiecePosition *moveFrom, PiecePosition
*moveTo )
{
  int ourRow, ourCol,theirRow,theirCol,row,col,runRow,runCol;
  int rowFirst = storage->playerColor == kRed ?
            0 : kBoardSize - 1;
  int rowLast = storage->playerColor == kRed ?
            kBoardSize - 1 : 0;
  int rowAdd = storage->playerColor == kRed ? 1 : -1;
  int bestUnknowns;
  int bestPath;
  int thisPath;

  UpdateDangerPossibilities( storage );

// USE SPY
  if ( FindPiece( storage, storage->theirColor, kMarshall,
                &theirRow, &theirCol )
       && FindPiece( storage, storage->playerColor, kSpy,
                &ourRow, &ourCol )
       && abs( theirRow - ourRow ) +
            abs( theirCol - ourCol ) == 1 ) {
    moveFrom->row = ourRow;
    moveFrom->col = ourCol;
    moveTo->row = theirRow;
    moveTo->col = theirCol;
    RETURN( "\pUSE SPY" );
  }

// DEFEND AGAINST SPY
  if (storage->their_pieces[kSpy] > 0) {
    if ( FindPiece( storage, storage->playerColor, kMarshall,
                &ourRow, &ourCol ) ) {
      int unknowns = CountAdjacentUnknownPieces( storage,
                    storage->theirColor, ourRow, ourCol );

      if ( unknowns ) {
        char todo = 0; // R = Run,A = Attack, O = Attack with Other
        int base_index = 0;
    Boolean canrun = FindSafeSquare( storage, ourRow, ourCol,
                                        &runRow, &runCol );
        if ( !canrun ) {
          base_index += 16;
        }
        if ( unknowns > 1 ) {
          base_index += 8;
        }

        for ( int d = 0; d < 4; d++ ) {
          int r = ourRow + dR[d];
          int c = ourCol + dC[d];
          int otherRow, otherCol;

          if ( IsOnBoard( r, c )
              && IsTheirPiece( storage, r, c )
              && IsUnknownPiece( storage, r, c ) ) {
            int index = base_index;
            if ( LowlyCanAttack( storage, r, c,
                          &otherRow, &otherCol ) ) {
              index += 4;
            }
            if ( CountAdjacentUnknownPieces( storage,
                      storage->theirColor, r, c ) > 0 ) {
              index += 2;
            }
            if ( IsMovedPiece( storage, r, c ) ) {
              index += 1;
            }

            if ( defend_spy_table[index] == 'A' ) { // Attack
              moveFrom->row = ourRow;
              moveFrom->col = ourCol;
              moveTo->row = r;
              moveTo->col = c;
              RETURN( "\pDEFEND AGAINST SPY 1" );
            } else if ( defend_spy_table[index] == 'O' ) {
```

```
Attack
                moveFrom->row = otherRow;
                moveFrom->col = otherCol;
                moveTo->row = r;
                moveTo->col = c;
                RETURN( "\pDEFEND AGAINST SPY 2" );
              }
            }
          }

          if ( canrun && OKMove( storage, ourRow, ourCol,
                                        runRow, runCol ) ) {
            moveFrom->row = ourRow;
            moveFrom->col = ourCol;
            moveTo->row = runRow;
            moveTo->col = runCol;
            RETURN( "\pDEFEND AGAINST SPY 3" );
          }
          // Give up! Next rule...
        }
      }
    }

// ATTACK WEAKER
  for ( row = rowFirst; 0 <= row && row < kBoardSize;
                        row += rowAdd ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsTheirPiece( storage, row, col ) ) {
        UInt32 enemy = storage-
>board[row][col].possibilities;
        UInt32 danger = GetDangerPossibilities( storage,
                                            row, col );

        int bestDir = -1;
        Boolean isBestRevealed = true;
        PieceRank bestRank = kUnknown;

        for ( int d = 0; d < 4; d++ ) {
          int r = row + dR[d];
          int c = col + dC[d];

    if ( IsOnBoard( r, c ) && IsOurPiece( storage, r, c ) )
{
            if ( !PossibilitiesCouldKill(
                      storage->board[r][c].rank, danger ) )
{
              if ( WillKillPossibilities(
                      storage->board[r][c].rank, enemy ) )
{
                Boolean thisRevealed =
                          IsRevealedPiece( storage, r, c );
                if ( isBestRevealed || !thisRevealed ) {
                  if ( bestDir == -1 ||
                  (storage->board[r][c].rank > bestRank) )
{
                    bestDir = d;
                    bestRank = storage->board[r][c].rank;
                    isBestRevealed = thisRevealed;
                  }
                }
              }
            }
          }
        }
        if ( bestDir != -1 ) {
          moveFrom->row = row + dR[bestDir];
          moveFrom->col = col + dC[bestDir];
          moveTo->row = row;
          moveTo->col = col;
          RETURN( "\pATTACK WEAKER" );
        }
      }
    }
  }
```

```
// EXPLORE ATTACK
  for ( int rnk = kScout; rnk >= kMarshall; rnk- ) {
    PieceRank rank = (PieceRank) rnk;
    if ( IsLowlyRank( rank ) ) {

      for ( row = rowLast; 0 <= row && row < kBoardSize;
                                    row -= rowAdd ) {
        for( col = 0; col < kBoardSize; col++ ) {
          if ( IsOurPiece( storage, row, col )
              && IsRankPiece( storage, row, col, rank ) ) {

            for ( int d = 0; d < 4; d++ ) {
              int r = row + dR[d];
              int c = col + dC[d];

              if ( IsOnBoard( r, c )
                && IsTheirPiece( storage, r, c )
                && IsRankPiece( storage, r, c, kUnknown ) )
{
                moveFrom->row = row;
                moveFrom->col = col;
                moveTo->row = r;
                moveTo->col = c;
                RETURN( "\pEXPLORE ATTACK" );
              }
            }
          }
        }
      }
    }

// RETREAT
  for ( row = rowLast; 0 <= row && row < kBoardSize;
                                row -= rowAdd ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsOurPiece( storage, row, col )
          && IsMovedPiece( storage, row, col ) ) {

        for ( int d = 0; d < 4; d++ ) {
          int r = row + dR[d];
          int c = col + dC[d];

          if ( IsOnBoard( r, c )
                && IsTheirPiece( storage, r, c )
                && WillPossibilitiesKill(
                        storage->board[r][c].possibilities,
                        storage->board[row][col].rank ) ) {
            bestPath = 1000;
            for ( int to_row = rowLast; 0 <= to_row &&
                    to_row < kBoardSize; to_row -= rowAdd )
{
        for( int to_col = 0; to_col < kBoardSize; to_col++ )
{
                thisPath = bestPath;
              if ( IsTheirPiece( storage, to_row, to_col )
                  && (IsRankPiece( storage, to_row, to_col,
                                        kUnknown )
                    || WillKillPossibilities(
                        storage->board[row][col].rank,
            storage->board[to_row][to_col].possibilities ))
            && FindSafePath( storage, false, true, row, col,
              to_row, to_col, &thisPath, &runRow, &runCol )
            && OKMove( storage, row, col, runRow, runCol ) )
{
                    bestPath = thisPath;
                    moveFrom->row = row;
                    moveFrom->col = col;
                    moveTo->row = runRow;
                    moveTo->col = runCol;
                  }
                }
              }
              if ( bestPath < 1000 ) {
                RETURN( "\pRETREAT" );
              }
            }
          }
        }
      }
    }
```

```
// SCOUT
  bestUnknowns = 0;

  for ( row = rowLast; 0 <= row && row < kBoardSize;
                                    row -= rowAdd ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsOurPiece( storage, row, col )
            && IsRankPiece( storage, row, col, kScout ) ) {
        for ( int d = 0; d < 4; d++ ) {
          int r = row + dR[d];
          int c = col + dC[d];

          while ( IsOnBoard( r, c ) &&
                    IsEmptySquare( storage, r, c ) ) {

            int knowns, unknowns;
            CountEnemies( storage, r, c, &knowns, &unknowns );
            if ( knowns == 0 && unknowns > bestUnknowns &&
                      OKMove( storage, row, col, r, c ) )
{
              bestUnknowns = unknowns;
              ourRow = row;
              ourCol = col;
              runRow = r;
              runCol = c;
            }
            r += dR[d];
            c += dC[d];
          }
        }
      }
    }
  }

  if ( bestUnknowns > 0 ) {
    moveFrom->row = ourRow;
    moveFrom->col = ourCol;
    moveTo->row = runRow;
    moveTo->col = runCol;
    RETURN( "\pSCOUT" );
  }
```

```
// ATTACK DISTANT

  bestPath = 1000;

  for ( row = rowFirst; 0 <= row && row < kBoardSize;
                          row += rowAdd ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsTheirPiece( storage, row, col ) ) {
        UInt32 possibilities =
            storage->board[row][col].possibilities;
        UInt32 danger =
            GetDangerPossibilities( storage, row, col );

        if ( (possibilities & ((1 << kBomb) | (1 <<
kMarshall))) !=
                                    ((1 << kBomb) | (1 <<
kMarshall)) ) {
          for ( int r = rowFirst; 0 <= r && r < kBoardSize;
                              r += rowAdd ) {
            for( int c = 0; c < kBoardSize; c++ ) {
              if ( IsOurPiece( storage, r, c ) ) {
                if ( WillKillPossibilities(
                storage->board[r][c].rank, possibilities ) )
{
                if ( storage->board[r][c].rank >= kCaptain ||
                      !PossibilitiesCouldKill(
                        storage->board[r][c].rank, danger ) )
{
                      thisPath = bestPath;
                if ( FindSafePath( storage, true, false, r, c,
                    row, col, &thisPath, &runRow, &runCol ) )
{
                if ( OKMove( storage, r, c, runRow, runCol ) ) {
                          bestPath = thisPath;
                          moveFrom->row = r;
                          moveFrom->col = c;
                          moveTo->row = runRow;
                          moveTo->col = runCol;
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }

  if ( bestPath < 1000 ) {
    RETURN( "\pATTACK DISTANT" );
  }
```

```
// EXPLORE DISTANT

  bestPath = 1000;

  for ( row = rowFirst; 0 <= row && row < kBoardSize;
                          row += rowAdd ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsTheirPiece( storage, row, col ) &&
            storage->board[row][col].rank == kUnknown ) {

        for ( int r = rowFirst; 0 <= r && r < kBoardSize;
                                  r += rowAdd ) {
          for( int c = 0; c < kBoardSize; c++ ) {
            if ( IsOurPiece( storage, r, c ) &&
                  IsLowlyPiece( storage, r, c ) ) {
              thisPath = bestPath;
              if ( FindSafePath( storage, false, true, r, c,
                  row, col, &thisPath, &runRow, &runCol ) )
{
                if ( OKMove( storage, r, c, runRow, runCol ) )
{
                      bestPath = thisPath;
                      moveFrom->row = r;
                      moveFrom->col = c;
                      moveTo->row = runRow;
                      moveTo->col = runCol;
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }

  if ( bestPath < 1000 ) {
    RETURN( "\pEXPLORE DISTANT" );
  }
```

```
// ATTACK KNOWN WITH SAME DISTANT

  bestPath = 1000;

  for ( row = rowFirst; 0 <= row && row < kBoardSize;
                                  row += rowAdd ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsTheirPiece( storage, row, col ) ) {
        UInt32 possibilities =
            storage->board[row][col].possibilities;

        if ( (possibilities & ((1 << kBomb) | (1 <<
kMarshall))) !=
                                    ((1 << kBomb) | (1 <<
kMarshall)) ) {
          for ( int r = rowFirst; 0 <= r && r < kBoardSize;
                                  r += rowAdd ) {
            for( int c = 0; c < kBoardSize; c++ ) {
              if ( IsOurPiece( storage, r, c ) ) {
                if ( WillKillOrSuicidePossibilities(
                      storage->board[r][c].rank,
possibilities ) ) {
                  thisPath = bestPath;
                if ( FindSafePath( storage, true, true, r, c,
                    row, col, &thisPath, &runRow, &runCol ) )
{
                if ( OKMove( storage, r, c, runRow, runCol ) )
{
                      bestPath = thisPath;
                      moveFrom->row = r;
```

```
                    moveFrom->col = c;
                    moveTo->row = runRow;
                    moveTo->col = runCol;
                  }
                }
              }
            }
          }
        }

      }
    }
  }

  if ( bestPath < 1000 ) {
    RETURN( "\pATTACK KNOWN WITH SAME DISTANT" );
  }

// FIND FLAG
// NYI

// MOVE FORWARD

  for ( row = rowLast; 0 <= row && row < kBoardSize; row -=
rowAdd ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsOurPiece( storage, row, col ) ) {
        PieceRank rank = storage->board[row][col].rank;
        if ( rank != kBomb && rank != kFlag ) {
          int r = row + rowAdd;
          if ( IsOnBoard( r, col ) && !IsOurPiece( storage,
r, col ) && OKMove( storage, row, col, r, col ) ) {
            moveFrom->row = row;
            moveFrom->col = col;
            moveTo->row = r;
            moveTo->col = col;
            RETURN( "\pMOVE FORWARD" );
          }
        }
      }
    }
  }

// MOVE

  for ( row = rowLast; 0 <= row && row < kBoardSize;
                              row -= rowAdd ) {
    for( col = 0; col < kBoardSize; col++ ) {
      if ( IsOurPiece( storage, row, col ) ) {
        PieceRank rank = storage->board[row][col].rank;
        if ( rank != kBomb && rank != kFlag ) {

          for ( int d = 0; d < 4; d++ ) {
            int r = row + dR[d];
            int c = col + dC[d];

            if ( IsOnBoard( r, c ) &&
                 !IsOurPiece( storage, r, c ) &&
                 OKMove( storage, row, col, r, c ) ) {
              moveFrom->row = row;
              moveFrom->col = col;
              moveTo->row = r;
              moveTo->col = c;
              RETURN( "\pMOVE" );
            }
          }
        }
      }
    }
  }

// RESIGN
  moveFrom->row = -1;
  moveFrom->col = -1;
  moveTo->row = -1;
  moveTo->col = -1;
  RETURN( "\pRESIGN" );

}
```

HandleOurMove

```
static void HandleOurMove( StoragePtr storage,
                           ReportYourMove ReportMove )
{
  PiecePosition moveFrom;
  PiecePosition moveTo;
  Boolean moveStrike;
  MoveResult moveResult;
  UnsignedWide start, finish;

  if ( storage->total_time > 10000000 ) { //Time to give up
    // Resign
    moveFrom.row = -1;
    moveFrom.col = -1;
    moveTo.row = -1;
    moveTo.col = -1;
  } else {
    FigureOutOurMove( storage, &moveFrom, &moveTo );
  }
  if ( IsOnBoard( moveTo.row, moveTo.col ) ) {
  moveStrike = storage->board[moveTo.row][moveTo.col].color !=
                                          kNoColor;
  } else {
    moveStrike = false;
  }

  Microseconds( &start );
  (*ReportMove)( &moveFrom, &moveTo, moveStrike, &moveResult
);
  Microseconds( &finish );
  storage->extra_time += finish.lo - start.lo;

  if ( moveResult.victory ) {        // We Win! :-)
        storage->victory = true;
  } else if ( !moveResult.legalMove ) { // We Lose! :-(
  } else {
    if ( moveStrike ) {
      storage->repeated_board_count = 0;
      Learn( storage, true, moveTo.row, moveTo.col,
             moveResult.rankOfDefender.thePieceRank );
      Learn( storage, false, moveFrom.row, moveFrom.col,
             moveResult.rankOfAttacker.thePieceRank );

      if ( moveResult.attackerRemoved &&
           moveResult.defenderRemoved ) {
        storage->board[moveFrom.row][moveFrom.col] =
            storage->blankSquare;
        storage->board[moveTo.row][moveTo.col] =
            storage->blankSquare;
      } else if ( moveResult.attackerRemoved ) {
        if ( storage-
>board[moveTo.row][moveTo.col].rank==kBomb ) {
          storage->board[moveFrom.row][moveFrom.col] =
              storage->blankSquare;
        } else {
          storage->board[moveFrom.row][moveFrom.col] =
              storage->board[moveTo.row][moveTo.col];
          storage->board[moveTo.row][moveTo.col] =
              storage->blankSquare;
        }
      } else {
        Assert( moveResult.defenderRemoved );
        storage->board[moveTo.row][moveTo.col] =
            storage->board[moveFrom.row][moveFrom.col];
        storage->board[moveFrom.row][moveFrom.col] =
            storage->blankSquare;
      }

    } else {
      if ( abs( moveTo.row - moveFrom.row ) +
           abs( moveTo.col - moveFrom.col ) > 1 ) {
        Assert( storage-
>board[moveFrom.row][moveFrom.col].rank ==
                                          kScout );
  Learn( storage, false, moveFrom.row, moveFrom.col, kScout);
      } else {
        Learn( storage, false, moveFrom.row, moveFrom.col,
                                    (PieceRank)kMoved );
      }
      storage->board[moveTo.row][moveTo.col] =
            storage->board[moveFrom.row][moveFrom.col];
      storage->board[moveFrom.row][moveFrom.col] =
```

```
            storage->blankSquare;
      }
      AppendRepeatedBoard( storage );
   }

   AssertValidBoard( storage );
}
```

MakeAMove

```
Boolean MakeAMove(
   void *privStorage,          /* 1MB of storage from PositionPieces */
   PlayerColor playerColor,    /* you play red or blue, with red playing first */
   GetOpponentMove *GetMove, /* callback used to find about opponents last move*/
   ReportYourMove *ReportMove /* callback used to make a move */
)
{
   StoragePtr storage = (StoragePtr) privStorage;
   UnsignedWide start, finish;

   storage->extra_time = 0;
   Microseconds( &start );

   if ( storage->do_getmove ) {
      HandleTheirMove( storage, *GetMove );
   }
   storage->do_getmove = true;

   HandleOurMove( storage,  *ReportMove );

   Microseconds( &finish );
   storage->total_time +=
         finish.lo - start.lo - storage->extra_time;

   return storage->victory;
}
```

CHALLENGE.H

```
#ifndef __LL_CHALLENGE__
#define __LL_CHALLENGE__

#ifdef __cplusplus
extern "C" {
#endif

#define kBoardSize 10

typedef enum { kUnknown=0,
   kMarshall=1,kGeneral,kColonel,kMajor,kCaptain,
   kLieutenant,kSergeant,kMiner,kScout,kSpy,
   kBomb,kFlag
} PieceRank;

typedef enum {kRed=1,kBlue=2} PlayerColor;

typedef struct PieceType {
   PieceRank  thePieceRank;      /* rank of a piece */
   PlayerColor thePieceColor;    /* color of a piece */
} PieceType;

typedef PieceType Board[kBoardSize][kBoardSize];
/* Used to provide test code with board configuration.  Red starts
   in rows 0..3, Blue starts in rows 6..9 */
/* Squares [4][2], [4][3], [4][6], [4][7] and
        [5][2], [5][3], [5][6], [5][7] are water and cannot
        be occupied */

typedef struct PiecePosition {
   long row;   /* 0..9 */
   long col;   /* 0..9 */
} PiecePosition;

typedef struct MoveResult {
   PieceType rankOfAttacker;
   /* after a strike, returns identity of attacker */
   PieceType rankOfDefender;
   /* after a strike, returns identity of defender */
   Boolean attackerRemoved;
   /* true after a strike against a piece of equal or greater rank,
      or against a bomb when the attacker is not a Miner */
   Boolean defenderRemoved;
   /* true after a strike by a piece of equal or greater rank,
      or against a bomb when the attacker is a Miner,
      or against a Marshall by a Spy */
   Boolean victory;
   /* true after a strike against the Flag */
   Boolean legalMove;
   /* true unless you
      - move into an occupied square, or
      - move or strike in a direction other than forward, backward, or sideways, or
      - move more than one square (except Scouts), or
      - move a Bomb or a Flag,
      - move into Water, or
      - strike a square not occupied by an opponent, or
      - make any other illegal move  */
} MoveResult;

void PositionPieces(
   void *privStorage,        /* 1MB of preinitialized storage for your use */
   PlayerColor playerColor, /* you play red or blue, with red playing first */
   Board *theBoard           /* provide the initial position of your pieces */
);

typedef void (*ReportYourMove)(
                             /* Callback to inform test code of move and
                                get results */
   PiecePosition *moveFrom, /* piece you are moving or using to strike */
   PiecePosition *moveTo,   /* destination square or square being struck */
   Boolean strike,          /* false indicates a move, true indicates a strike */
   MoveResult *results      /* returns identity of struck piece and other info */
);

typedef void (*GetOpponentMove)(
                          /* Callback to get results of opponents last move */
   PiecePosition *moveFrom,    /* piece opponent moved or used to strike */
   PiecePosition *moveTo,      /* destination square or square struck */
   Boolean *strike,            /* false indicates a move, true indicates a strike */
   MoveResult *results         /* returns identity of struck piece and other info */
);

Boolean MakeAMove(
   void *privStorage,          /* 1MB of storage from PositionPieces */
   PlayerColor playerColor,    /* you play red or blue, with red playing first */
   GetOpponentMove *GetMove,   /* callback used to find about opponents last move
*/
   ReportYourMove *ReportMove /* callback used to make a move */
);

#ifdef __cplusplus
}
#endif

#endif
```

Want to know what products are available for MacOS development? Check out Developer Depot™ <http://www.devdepot.com>

DIALOG BOX

by Robert Hettinga, Boston

What Is Cryptography Good For, Anyway?

A look at why the market demands cryptography, because it makes electronic business more efficient

Cryptography allows you to do business with, work with, and trust people you don't know. It can also save a lot of time, programming, and machine resources, and thus, money.

Many people who talk about cryptography talk about it in political terms. We hear people talk about civil liberties, about freedom of speech, the right to bear "arms" (crypto is classified as a munition) or even freedom from having the government quarter "troops", in the form of key escrow authorities, on our hard drives. Frankly I've gotten tired of all the politics. Cryptography, like any other technology, is value neutral. Just like any form of progress, cryptography won't be adopted unless it makes our lives better. And that, I assure you, is what it is going to do.

I've talked extensively in speeches, on the net, and in articles like this one, about financial cryptography and how it's going to change the world, not by making our transactions invisible to big brother, but by forcing profit and loss responsibility down onto smaller and smaller organizations, and, eventually, to applications and microprocessors themselves. I joke about the day when, instead of a credit card association and a bank loaning us money for lunch, it will be a syndicate of individual "bond-bots" each taking a small piece of what could be called a personal digital bearer bond issue for that lunch, all based on our reputation and ability to repay.

I talk about routers which would move information around the net by charging minuscule bits of picocash, buying bandwidth low and selling it high — sender pays — in an instantaneously settled auction market for packet switching. Good bye to peering fights, NAPs, and the emerging hierarchy of super-routers and high-capacity backbones. Since each router makes money instead of costs money, it behooves routers to be connected to several other routers, creating a geodesic, instead of hierarchical, Internet. When a router saves enough money out of operations, it could buy a copy of itself. This whole idea of a self-organizing ecology of microeconomic entities makes a lot of people yell at me, particularly those who've spent their careers building bigger and bigger systems, but, then, as my friend Rodney Thayer says, "you're only as good as the people who yell at you". Since I've had some pretty clueful people yell at me, I must be on to something.

Financial Basics

Let's start with a little finance. It will help us understand things a little better.

One of the pervasive notions in the economy is that of the book entry. Modern double-entry bookkeeping is about keeping debits and credits in a database. Most of our transaction systems are about sending these debits and credits over wires: credit cards, checks, and almost all transaction settlement in the capital markets are all done with book-entry settlement. Try asking your broker for physical delivery of a stock certificate sometime, and you'll see what I mean.

We have book-entry settlement now because when telegraphy was invented, you couldn't send a bearer certificate, like those old fashioned bonds with rows of coupons on the bottom — or paper money — down a wire. We could send only stuff like "I'm debiting this amount from my account, please credit yours by the same."

Robert Hettinga <rah@shipwright.com> is a financial cryptography industry pundit. He started several e-mail lists, a web site, a monthly luncheon group, and even an annual conference in Anguilla, all to talk about financial cryptography. See the e$ web site <http://www.shipwright.com/> for more information about his various services.

One problem with book-entries is that you have to trust who sent them to you. This is usually done with access control lists and private proprietary networks. "Clubs" if you will, with a list of members, all biometrically identified (the SEC doesn't take fingerprints for fun), and strict rules for doing things with other members of the club. Break the rules and get thrown out of the club, or worse.

This is different from bearer certificates. With a bearer certificate, you can tell by inspection that the certificate is valid, and, if you know the public reputation of the person or company who issued the certificate, you can decide whether or not to trust them. For example, you can pretty much tell that a dollar bill is genuine by inspection, and you can trust that a modern dollar bill is worth something, but that a Confederate dollar isn't, simply by knowing the public reputation of the issuer.

Another problem with book-entries is that because they are basically unsecure transactions sent down a secure network, we have to have some kind of sanction to prevent fraud. By "sanction" we usually mean violence, usually "sold" by a nation-state of some kind. Nick Leeson, who recently brought down Barings Bank, was sent to jail for making the wrong book-entry. (They tried in Singapore but he escaped to Germany and was extradited to Britain.) Of course, if you send a book-entry to a machine in St. Louis, but you're in Kampala, there's a problem. The answer, of course, is a global government and police force. Right. Go look up Occam's Razor in the dictionary for an answer to that one...

With a bearer certificate, you can shun people who cheat you, which, in some ways, is better than violent sanction. It's certainly cheaper. Ask the Amish how well shunning works as social control. In a financial market, shunning is economic death. Nobody will do business with you.

The very biggest problem with digital book-entries is tha they cost so much, and I don't mean just in paying taxes t support police. I mean in computer processing and storage. No only must we keep lists of who can do what to whom and fo how much, we also have to keep records of what we did wit anybody else, so we can bust them for doing something wron to us later. Also, for every transaction regime, there must be trusted third party, usually called a clearinghouse, who ha records of what everybody did to everybody else. On a typica credit card transaction, for instance, you, me, your bank, m bank, and the credit card company all have a record of the lunc I bought from you. We aren't even talking about the check I send to my bank monthly to actually settle my credit card transactions.

We also won't talk about the fact that anyone who scored high enough on a civil service test and now works at the Financial Crimes Enforcement Network (FinCEN) has the right to see those transactions. That's because, again, heretical as it is to the civil liberties folks, cryptography is not really about privacy. It's about economic efficiency and progress.

Creating Digital Bearer Certificates

How can we send a bearer certificate down a wire? Because we can now create digital bearer certificates using the blind signature algorithm developed by David Chaum, the founder of DigiCash. Using this algorithm I can create a unique cryptographic object which has value in the same way that a dollar bill is a unique printed object having value. Of course, those cryptographic objects can be moved down a wire.

Moore's Law dictates a more geodesic network by automating switching and dropping its cost in half every 18 months. It also allows us to pay for that switching very efficiently,

by allowing us to automate and manipulate blind signatures and other cryptographic algorithms very cheaply. We, or better, our machines, can issue and spend these bearer certificates of any transaction size, from trillions of dollars to trillionths of a cent, all without keeping transaction records or access lists.

Pull the change out of your pocket and look at it. Do you remember where each and every coin came from? Do you care? When you spend them in a soda machine, does it care? No. Imagine a world where the soda machine or the Internet takes VISA. And, no, I don't mean of big brother, either. Imagine if the whole net cleared on a 90 day float time, at 18% interest... It's absurd.

But, of course, I'm still not really here to talk about financial cryptography. I'm here to talk about access control.

Controlling Access with Bearer Certificates

Well, suppose you had some code you wanted to limit access to, say the SubWoofer source code, or maybe beta version of your software. Suppose, instead of creating and managing a list of developers and what they can see, or even passing around an easily compromised password, you just handed each of them a unique cryptographic object. A ticket, if you will. People could download the package, but only if they cashed in a ticket for it.

The neat thing about this idea is that you don't care who shows up with a ticket, because the tickets are unique and unreplicable. They have value, the value of one download of the SubWoofer source. You can e-mail them out, and if the person who receives the ticket doesn't use it and gives it to someone else, you still have issued only a finite number of copies of the code. Anyone who shows up with a duplicate ticket doesn't get the package. If you're really draconian, Chaum's protocol lets you take the "double-spent" ticket, compare it with the ticket you have taken in already, and identify the key which duplicated the ticket. No access control lists, but you still have to hang onto the tickets which have been turned in.

There are other problems, too. The above actually involves setting up a Chaumian mint and having patented, signature-blinding walletware. Unfortunately, DigiCash, like Chaum before them, have been playing dog in the manger with the patent and are not licensing it to people who could actually make some use of it. There has always been a problem of mistaken identity at DigiCash. First they thought they were CitiCorp, then they thought they were Microsoft, now they think they're VISA. Someday they'll wake up and realize they're cryptographers, or possibly Dolby and Co. (the audio technology people), and we'll all be better off for it.

Fortunately, there is an almost equivalent way to get the same result with minor modifications to an existing, public code base: PGP. It should be possible to do the following neat hack, a sort of a poor man's certification authority. Actually, we're creating something more important than a hierarchical "authority", we're creating a small, geodesic, "web" of trust.

First, create a private PGP key which authorizes people to access the package. (PGP allows you to generate multiple private keys. Just create one for this particular "permission".) Then, using that key, digitally sign the public key of people who you want to have access, and send their signed keys to them. Now, create a quick and dirty browser plugin to hold a copy of the person's signed public key. (This also can then be used for other kinds of access signatures later.) Actually, you might as well put their signed key into the plugin and send it to them that way, since they won't have the plugin the first time around, anyway. Next, put a CGI on your webserver which reads the key in the plugin, and checks to see if it's signed by the right key. Again, this is a single key, an access control "list" which will always have one "record". Well, I suppose you might have two or three people, so you could have them each generate a special purpose key pair of their own, and store the public keys in the list of authorized signatures. Only the person who owns the key in the plugin can make the plugin work.

To use it, someone puts the plugin where their browser wants to see it, and goes to the URL you told them to. The CGI checks the "ticket pocket" plugin and sees if their signature is signed by the key which authorizes access. If not, they go to a page which tells them how to get permission. If they do have permission, then they just see the download page automatically. They can download as many copies of the package as they want, and whether they hand it around is covered, hopefully, by an NDA of some kind.

By the way, when someone talks about cryptographically "watermarking" an application, remember that all this does is tell where the code was stolen from, not who stole it. Clearly, this makes "watermarking" things a waste of time.

Anyway, once we've built the pieces, we can use this technology for anything we want to control access to. No passwords, no users accounts, no group list — hardly any management at all. In fact, if everyone has the plugin already, all the authorizing person has to do is to download someone's public key off a keyserver somewhere and mail them a signed copy of it.

Cryptography is Easier than Bookkeeping

So now you know why cryptography is so cool, and, most especially, efficient. You don't need vulnerable and expensive databases, with probably secure but potentially unreliable session "pipes" linking them (SSL and SET for example), all to just move permissions, or decision rules, or abstractions of value — like money — around the net. Anytime you're confronted with a large and volatile database, especially if it requires another large list of people to have permission to change data in that list, ask yourself if you could do it all much better by creating cryptographic objects and moving them around instead of database entries.

As our ticket and certification web examples show, cryptography usually offers a better way to do it.

Occam's Cryptography, if you will. Cryptography is a weapon, remember? MT

Visit MacTech® Magazine's Web site!
http://www.mactech.com

by Daniel W. Rickey, Manitoba Cancer Treatment and Research Foundation

Getting a Speed Boost with Fixed-Point Math

A little assembly goes a long way to improving code execution speed

Motivation

The recent releases of the Metrowerks compilers allow Pascal programmers to include assembly language in their programs as functions and procedures. A built-in assembler is a very powerful tool because many of the features of the Pascal environment are available in the assembler. For example, compiler directives enable a routine to contain both 68k and PowerPC assembly code. In addition, the integrated environment provides seamless source-level debugging of the assembly and Pascal code. (C programmers can gain the same benefits, but they must write complete functions to use the built-in assembler.)

One of the biggest advantages of assembler is that it allows the programmer to use processor instructions that are difficult or impossible to access from a high-level language. Because of this ability, the programmer may have the option of using a more efficient algorithm when coding in assembler. For these reasons, the assembler is an excellent tool for optimizing time-critical routines. This article shows how the built-in assembler can be used to optimize some simple fixed-point math routines.

Fixed-Point Math

In many imaging applications, such as texture mapping or ray casting of three-dimensional medical data sets (Herman 1993), approximately 10^5 to 10^6 pixel values are mapped to the screen. For interactive viewing, we need to calculate at least 10^5 position and intensity values each second. These calculations can be performed with floating-point math and the resulting values rounded to integer format for display. Unfortunately, the round function is very slow, particularly on 68k machines lacking a 68882 co-processor, e.g. machines using a 68LC040. In fact, all floating-point operations on machines lacking a co-processor are very slow. Although the PowerPC processors offer excellent floating-point performance they are relatively slow at rounding because this operation requires a move to and from main memory. To overcome these speed limitations, programmers can use fixed-point calculations instead of floating-point operations. On a 68k processor, rounding a fixed-point number can be 50 times faster than rounding a floating-point number. If a co-processor is not present, other operations, such as multiplication, will be much faster using fixed-point math. On PowerPC-based machines the fixed-point round is about 3 times faster than a floating-point round. The speed advantage gives strong motivation to use fixed-point math.

Fixed-point math is surprisingly easy to use. On Macintosh a fixed-point number is 32 bits long, that is Fixed = LongInt;. The low-order 16 bits contain the fractional part of the number and the high-order 16 bits contain the integer portion of the number. Addition and subtraction are handled as they normally would be: by simply adding and subtracting the numbers. Multiplication, division and type conversions must be performed using function calls. The Toolbox functions for performing these operations are:

```
FUNCTION FixMul(a, b: Fixed): Fixed;
FUNCTION FixDiv(a, b: Fixed): Fixed;

FUNCTION Fix2X(x: Fixed): double_t;
FUNCTION X2Fix(x: double_t): Fixed;

FUNCTION Long2Fix(x: LongInt): Fixed;
FUNCTION Fix2Long(x: Fixed): LongInt;
FUNCTION FixRound(x: Fixed): Integer;
```

Daniel W. Rickey is a medical imaging physicist with a background in Doppler ultrasound. He started programming a shareware 3-D image display program (MacCubeView) on a Mac Plus and has since decided that a PowerMac is faster. He can be reached at the Department of Medical Physics, 100 Olivia St., Winnipeg, Manitoba, R3E 0V9 Canada. daniel@kaon.mctrf.mb.ca.

Because the extended variable type is not supported on a PowerPC, the original Fix2X and X2Fix functions were updated to replace the Extended type with double_t, which is equal to a Double on a PowerPC and an Extended on a 68k processor.

Goals

The primary motivation for using fixed-point calculations is speed. Unfortunately, neither the 68k nor PowerPC Toolbox routines are well optimized. In fact, these routines were not PowerPC native until system 7.5.3. In this article we will concentrate on optimizations of the multiply, divide, and round functions as they are used most often. For this work we will use the built-in assembler, available in the Metrowerks Pascal Compiler, to write 68k and PowerPC code.

In addition to being fast, we require our functions to be well behaved. Because fixed-point numbers must lie between 32767.9999 and -32768, there exists a possibility of overflow during multiplication and division. If an overflow occurs, the function returns a large value with the appropriate sign. In the case of multiplication, an overflow condition behaves similar to the Toolbox routine: $7FFE0000 is returned when both arguments have the same sign and -$7FFE0000 is returned when the arguments have different signs. The largest possible fixed-point value is not returned as this might cause an overflow in the calling code. Note that these examples differ from the 68k assembler code of Lebedev (1994), which did not deal with overflow conditions.

Fixed-Point Round

The algorithm for rounding a fixed-point number is very simple: one half is added to the number and then the result is bit-shifted right by 16 bits. Shown in Listing 1 is the function for rounding a fixed-point number. A normal function declaration is used followed by the asm directive. Compiler directives are used to determine whether 680x0 or PowerPC instructions are used. Note that when compiled for a 68000-based machine, the Toolbox routine is used. This is because we make use of instructions that are available only on 68020 and newer processors.

680x0

Those unfortunate souls who program in C should note that the conventions for passing parameters to and from a 68k function differ from the Pascal code shown here. The function begins with the fralloc directive, which creates a stack frame using link a6, #$0. Once x is loaded into register d0, it is added to $7FFF, which, to maintain consistency with the PowerPC code shown below, is a tad less than one half ($8000). An algebraic shift is used to sign extend the result. Note that the "immediate" form of the asr bit-shift instruction can only shift a maximum of 8 bits, thus two instructions are required. The result is placed on the stack in the location allocated by the caller, i.e. $C(a6). Note that (a6) is the previous value of a6; $4(a6) is the return address; $8(a6) is the value of x; and $C(a6) is the space allocated for the returned value. The frfree directive simply undoes what fralloc did: it emits unlk a6. After executing this instruction, a6 will contain the previous value of a6 and a7 will point to the return address. Finally the function executes the rtd instruction to return to the caller by loading the return address into the program counter. This instruction also causes the stack pointer to point to the returned value thereby de-allocating the stack space originally allocated for x.

PowerPC

Compared to the 680x0, assembly language on the PowerPC processor is much more civilized. For example, we don't have to sort out the gruesome details of address modes and stack pointers because parameters are passed in registers and the same conventions are used for Pascal and C. In addition, there are only a few simple addressing modes compared to the fourteen available on the 680x0.

As described by Evans (1996) and Kacmarcik (1995), volatile registers r3 through r10 are used to pass parameters to the function. The PowerPC implementation of the MyFixRound function illustrates this: x is passed to the function in register r3 and the function result is returned in register r3. The PowerPC assembler differs from the 68k assembler because a stack frame is not required as long as the number of local variables is kept small. The first instruction addic (add immediate with carry) adds the sign-extended 16-bit $7FFF to r3 and places the result in r3. Because of the sign extension, we cannot add the exact one-half value of $8000. However, this shortcoming should not make any difference to real-world applications. The srawi (shift right algebraic word immediate) instruction is used to perform an algebraic shift right by 16 bits. The standard blr (branch to link register) instruction ends the function. (The link register was loaded with the return address when the function was called.) It should be noted that in PowerPC assembly language there are many extended mnemonics that the assembler converts into valid instructions. However, the Metrowerks compiler may not recognize all of them.

Listing 1: MyFixRound

```
Function MyFixRound(x :Fixed): LongInt;
{$IFC NOT POWERPC} {$IFC NOT OPTION(MC68020)}
INLINE $A840; {on a 68000 use the Toolbox routine}
{$ENDC} {$ENDC}


Function MyFixRound(x :Fixed): LongInt;
asm;
Begin
{$IFC OPTION(MC68020)}
  fralloc                 {create an a6 stack frame}
  move.l x, d0;           {move x into register d0}
  add.l  #$7FFF, d0;      {add 1/2 to x}
  asr.l  #$08, d0;        {shift 32-bit word to right by 8 bits}
  asr.l  #$08, d0;        {shift 32-bit word to right by 8 bits}

  move.l d0, $000C(a6);   {place the result on the stack}
  frfree;                 {free the a6 stack frame}
  rtd  #$0004;            {return and de-allocate parameter}
{$ENDC}
{$IFC POWERPC}
  addic  r3, r3, $7FFF;   {add 1/2 to x}
  srawi  r3, r3, 16;      {shift word to right by 16 bits}
  blr;                    {return; results are returned in r3}
{$ENDC}

End; {MyFixRound}
```

FIXED-POINT MULTIPLY

The algorithm for multiplying two fixed-point values is also simple: multiply the two values together and then bit-shift the result to the right by 16 bits. Shown in Listing 2 is the function for multiplying two fixed-point numbers. The basic structures of the 680x0 and PowerPC functions are similar to those shown in Listing 1.

680x0

This function illustrates the use of a processor instruction (muls.l) that is not available from Pascal. The muls.l instruction multiplies two signed 32-bit numbers together to give a 64-bit result, which is contained in two registers. After performing the multiplication, the routine ensures that the result is smaller than the largest acceptable fixed-point value, i.e. the high word of the result must be smaller than $7FFF. If the result is within range, the bit shift right is performed. Because the result is contained in two registers, the bit-shift operation requires a couple of steps: 1) d1 is shifted right by 16 bits; 2) d2 is algebraically shifted left by 16 bits; and, 3) the high order two bytes of d2 are moved into d1. Pairs of instructions are used to perform the 16-bit shifts. In the case of an overflow or out-of-range result, the function returns a large number whose sign depends on the signs of the two arguments. Note that because two parameters are passed to this function, the result is placed in a different location on the stack than for the MyFixRound function, i.e. $10(a6) versus $0C(a6). In addition, the rtd instruction removes two four-byte parameters from the stack.

PowerPC

Performing a 32-bit by 32-bit multiply on PowerPC processor requires two instructions: mulhw calculates the high-order 32 bits, and mullw the low-order 32 bits. After performing the multiplication, the cmpwi instruction is used to check if the result is smaller than the largest acceptable fixed-point value. Note that cmpwi is an extended form of the cmpi (compare immediate) instruction. If the result is in range, a technique similar to the 680x0 example is used to combine the results stored in registers r5 and r6. However, only two instructions are needed on the PowerPC: the rlwinm (rotate left word immediate then AND with mask) instruction is used to bit shift the low-order 32 bits of r5 and the rlwimi (rotate left word immediate then mask insert) is used to move the lower 16 bits of r6 into the top 16 bits of r3. Out-of-range values are handled as in the 68k code.

Visit MacTech® Magazine's Web site!
http://www.mactech.com

Listing 2: MyFixMu

```
Function MyFixMul(x, y :Fixed):Fixed;
{$IFC NOT POWERPC} {$IFC NOT OPTION(MC68020)}
INLINE $A868;   {on a 68000 use the Toolbox routine}
{$ENDC} {$ENDC}

Function MyFixMul(x, y :Fixed):Fixed;
asm;
Begin
{$IFC OPTION(MC68020)}
  fralloc       {create an a6 stack frame}
  move.l x, d0; {move x and y values into registers}
  move.l y, d1;

  muls.l d0, d2:d1;         {do the multiplication d2:d1 := d0*d1}
  cmpi.l #$00007FFF, d2;    {check if the result is larger than the}
  bge Overflow;             {maximum allowed for a fixed-point number}

  cmpi.l #-$00007FFF, d2;   {check if the result is less than the}
  ble Overflow;             {minimum allowed for a fixed-point number}

  lsr.l  #08, d1;       {want the high-order 2 bytes of d1; shift right 16 bits}
  lsr.l  #08, d1;       {can only bit shift a maximum of 8 bits}

  asl.l  #08, d2;       {want the low-order 2 bytes of d2; shift left 16 bits}
  asl.l  #08, d2;       {use an algebraic shift to maintain the sign bit}

  move.w d1, d2;        {combine d2 and d1}

  bra Finished;

Overflow:
  move.l #$7FFE0000, d2;    {return a large number}

  tst.l  x;                 {compare x to zero to determine if it is negative}
  bge CheckY;               {if positive jump to CheckY:}
  neg.l  d2;                {multiply the large number by -1}

CheckY:
  tst.l  y;                 {compare y to zero to determine if it is negative}
  bge Finished;             {if positive jump to Finished:}
  neg.l  d2;                {multiply the large number by -1}

Finished:
  move.l d2, $0010(a6);     {place the result d2 on the stack}
  frfree;                   {free the a6 stack frame}
  rtd #$0008;               {return and de-allocate parameters}
{$ENDC} {68020}

{$IFC POWERPC}
  mulhw  r6, r3, r4;        {find x * y and place high-order 32 bits into r6}
  mullw  r5, r3, r4;        {find x * y and place low-order 32 bits into r5}

  cmpwi  r6, $7FFF;         {check if high-order value is larger than the}
  bge OverFlow;             {maximum allowed for a fixed-point number}

  cmpwi  r6, -$7FFF;        {check if high-order value is less than the}}
  ble OverFlow;             {minimum allowed for a fixed-point number}

  rlwinm r3, r5, 16, 0, 31; {shift r5 right by 16 bits}
  rlwimi r3, r6, 16, 0, 15; {move low 16 bits of r6 into top 16 bits of r3}
  blr;                      {return; result is returned in r3}

OverFlow:                   {have an overflow condition}
  lis r5, $7FFE;            {load r5 with $7FFE0000}

  cmpwi  r3, $0;            {compare to 0 to determine if x is negative}
  bge CheckY;               {if positive then go on to check sign of y}
  neg r5, r5;               {multiply r5 by -1}

CheckY:
  cmpwi  r4, $0;            {compare to 0 to determine if y is negative}
  bge Finished;             {we're done}
  neg r5, r5;               {multiply r5 by -1}

Finished:
  mr r3, r5;                {copy value from r5 into r3}
  blr;                      {return; results are returned in r3}
{$ENDC} {POWERPC}
End; {MyFixMul}
```

If you want to be in the know, then you need every article published in the first 12 years of MacTech® Magazine and Apple's develop™ issues 1-29!

...in THINK Reference™ format!

So hurry, pick up the phone, fire up the e-mail, launch that fax machine, or simply drop by our web site and order yourself this new release of the MacTech® CD-ROM.

- Almost 1600 articles from all 139 issues of MacTech Magazine (1984-1996)
- Includes Apple develop issues 1-29
- Improved hypertext, improved indices, and a new THINK Reference Viewer — for lightning quick access!
- New hyperlinks between articles
- 100+ MB of source code — use them in your applications, with no royalties!
- Full version of THINK Reference — the original online guide to Inside Macintosh, Vols. I-VI
- 80MB of FrameWorks/SFA archives and the most complete set of FrameWorks archives known
- Sprocket™! MacTech's tiny framework that compiles quickly and supports System 7.5 features
- The best threads from the Mac programmer newsgroups plus thousands of notes, tips, snippets, and gotchas
- Popular tools that Mac programmers use to increase their productivity and much more!

Developer DEPOT™

Web Site: http//www.devdepot.com • E-mail: orders@devdepot.com

Phone: 800-MACDEV-1 • Outside the U.S. & Canada: 805-494-9797 • Fax: 805-494-9798

MacTech is a registered trademark of Xplain Corporation. MacDev-1, THINK Reference, Developer Depot, Sprocket, JavaTech, WebTech, BeTech, and the MacTutorMan are trademarks of Xplain Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders.

FIXED-POINT DIVIDE

The algorithm for dividing two fixed-point numbers is to bit shift the numerator to the left by 16 bits and then perform the division. Shown in Listing 3 is the function for dividing two fixed-point numbers. The basic structures of the 680x0 and PowerPC functions are similar to those shown in Listing 2. However, two function interfaces are used, one for PowerPC and the other for 680x0.

680x0

This function also uses a processor instruction (**divs.l**) that is not available from Pascal. The **divs.l** instruction divides a 64-bit value by a 32-bit value to give a 32-bit result. Before the division is performed, two things are done: 1) the denominator is checked for a divide-by-zero case; and 2) the numerator is split with the low-order 16 bits placed into the high word of **d0** and the high-order 16 bits placed into the low word of **d1**. An algebraic shift is used to maintain the sign bits in **d1**. Once the division is complete, the **bvc** instruction is used to check for an overflow condition, in which case a large positive or negative number is returned. The result is placed on the stack and the function exited in the same manner as for **MyFixMul**.

PowerPC

Unfortunately, the PowerPC instruction set does not provide an integer 64 by 32-bit divide. Because of this limitation, I chose to use the floating-point unit to perform the calculations. Notice that the function parameters are declared as **Real** instead of **Fixed**. Instead of bit-shifting the numerator, we perform the equivalent operation of multiplying by 65536. The Metrowerks assembler allows a real number to be specified directly in the assembler code as is illustrated by loading **fp4** with 65536. The division is then performed with the **fdivs** instruction. Notice that we use single-precision multiplication and division instructions, which are faster than their double-precision counterparts. The function then uses **fctiw** to convert the real number to an integer word, which is stored in the low-order word of **fp3**. This instruction also takes care of any out-of-range conditions for us. Unfortunately there is no instruction for transferring data between general-purpose and floating-point registers. Thus we are forced to store the contents of **fp3** in a temporary variable in memory. The integer value can then be loaded into register **r3** before the function is exited.

Listing 3: MyFixDiv

```
{$IFC POWERPC}
Function MyFixDiv(x, y: Real): Fixed;
{$ELSEC}
Function MyFixDiv(x, y: Fixed): Fixed;
{$ENDC}
{$IFC NOT POWERPC} {$IFC NOT OPTION(MC68020)}
INLINE $A84D;          {on 68000 use Toolbox routine}
{$ENDC} {$ENDC}

{$IFC OPTION(MC68020)}
Function MyFixDiv(x, y: Fixed): Fixed;
asm;
Begin
  fralloc;             {create an a6 stack frame}
  move.l y, d2;        {d2 contains denominator}
  move.l x, d0;        {d0 contains numerator}

  tst.l  d2;           {compare denominator to zero}
  beq  Overflow;       {division by zero gives an overflow}

DoDivision:
  move.l d0, d1;       {d1 to contain upper 16 bits of numerator}

  asr.l  #08, d1;      {algebraic shift right 16 so the low word of d1}
  asr.l  #08, d1;      {contains the upper 16 bits of the numerator}

  lsl.l  #08, d0;      {bit shift left 16 so the high word of d0 contains}
  lsl.l  #08, d0;      {the lower 16 bits of the numerator}

  divs.l d2, d1:d0;    {d0 := d1:d0/d2; perform a 64 by 32-bit division}
  bvc  Finished;       {check for overflow; go to Finished: if there was
none}

Overflow:
  move.l #$7FFE0000, d0; {return a large number}

  tst.l  x;            {compare x to zero to check if it is negative}
  bge  CheckY;         {if positive jump to CheckY:}
  neg.l  d0;           {make the large number negative}

CheckY:
  tst.l  y;            {compare y to zero to determine if it is negative}
  bge  Finished;       {if positive jump to Finished:}
  neg.l  d0;           {multiply the large number by -1}

Finished:
  move.l d0, $0010(a6); {place the result d0 on the stack}
  frfree;              {free the a6 stack frame}
  rtd  #$0008;         {return and de-allocate parameters}
End;
{$ENDC}  {68020}

{$IFC POWERPC}
Function MyFixDiv(x, y: Real): Fixed;
asm;

Var
  tempStorage:Record
    highLong :LongInt;
    lowLong  :LongInt;
  End;

Begin
  lfs  fp4, 65536.0; {load 2^16 into fp4}
  fmuls  fp3, fp1, fp4;  {multiply numerator by 2^16}
  fdivs  fp3, fp3, fp2;  {do the division}

  fctiw  fp3, fp3;              {convert to an integer}
  stfd fp3, tempStorage;        {store result in temp memory}
  lwz  r3, tempStorage.lowLong; {load 4-byte integer into r3}
  blr;                          {return; result is returned in r3}
End;
{$ENDC}
```

BENCHMARKS

To test the speed increase given by these functions, operations were performed on 4096-element arrays (see **TestFixedPointMath.p** for the details). For these measurements, the routines were compared to the Toolbox routines on a PowerPC 7200/90 with an L2 cache and an LC 475 with a 33 MHz 68LC040. Each assembly routine and its Toolbox counterpart was executed 4096000 times and the elapsed time determined using **TickCount**. The elapsed times ranged from 1.17 seconds and up. From the speed increases given in Table 1, it can be seen that even with the PowerPC-native fixed-point routines built into System 7.5.3, our assembly language routines deliver a respectable increase in performance. As expected, our routines are much faster than the non-native routines in System 7.5.2. In addition, our 68k routines are substantially faster than the Toolbox routines in emulation mode or on the genuine 68LC040 processor.

	Speed Increase (%)		
	Fixed Round	Fixed Multiply	Fixed Divide
7200/90 System 7.5.3	56	18	49
7200/90 System 7.5.2	49	392	760
7200/90 68k emulation	159	71	130
LC 475 System 7.5.3	178	137	369

Table 1. *Percentage speed increase obtained using the assembly language routines. A 100% speed increase means that our routine ran in half the time required for the Toolbox routine.*

Discussion

In this article, I have shown that assembly-language routines are faster than the Toolbox routines. However, as described by Kacmarcik (1995), the actual increase in performance on PowerPC machines will depend on how the instructions are scheduled and how the data interact with the processor and L2 caches. The multiply and divide instructions use the majority of time spent in the routine, thus the removal or addition of a few fast instructions should not make much difference. The division function on the PowerPC used floating-point instructions and incorporated a relatively slow floating-point round. Although it might be advantageous to implement an integer algorithm, division is always slow compared to addition and multiplication and should be avoided. In the future, a significant increase in performance should be realized when the 64-bit PowerPC 620 processor is used in the Macintosh and the 64-bit multiply and divide instructions are performed with one integer instruction.

It is hoped that this article illustrates that a built-in assembler is an excellent tool for optimizing time-critical, bottle-neck routines. Although the functions shown here are unlikely to be the fastest implementations, they illustrate the capabilities and advantages of including assembler routines directly into Pascal programs. For example, the functions used some instructions that are not accessible from Pascal. Therefore, the ease with which assembler is included and debugged in a modern programming environment makes us question the philosophy that assembler code should only be used as a last resort.

Bibliography and References

1. Apple Computer. "Mathematical and Logical Utilities". *Inside Macintosh Operating: System Utilities*, pp. 3-11–3-49. Addison Wesley, 1994.
2. Evans, Dave. "Balance of Power: Introducing PowerPC Assembly Language." *develop*, pp. 23–28, Issue 21 March 1995.
3. Herman, Gabor T. "3D Display: A Survey From Theory to Applications". *Computerized Medical Imaging and Graphics* pp. 231–242, Volume 17, Issue 4/5 1993.
4. Hoxey, Steve, Faraydon Karim, Bill Hay and Hank Warren (editors) *The PowerPC Compiler Writer's Guide*. Wartham Associates, 1995 ISBN 0-9649654-0-2.
5. Kacmarcik, Gary. *Optimizing PowerPC Code: Programming the PowerPC Chip in Assembly Language*. Addison Wesley, 1995 ISBN 0-201-40839-2.
6. Lebedev, Alexei. "Fixed Point Math For Speed Freaks". *MacTech Magazine* (formerly *MacTutor*) 10:3 March 1994.
7. Motorola. *MOTOROLA M68000 FAMILY Programmer's Reference Manual*. 1992. (Available online from Motorola High Performance Embedded Systems Division, pirs.aus.sps.mot.com/docs/pubs).
8. *PowerPC Microprocessor Family: The Programming Environments.*, MPCFPE/AD (Motorola Order Number) and MPRPPCFPE-01 (IBM Order Number, available online from IBM Microelectronics, www.chips.ibm.com).
9. *PowerPC 601 RISC Microprocessor User's Manual*, Rev 1 MPC601UM/AD (Motorola Order Number) and 52G7484/(MPR601UMU-02) (IBM Order Number).

Help MacTech Work

Here at *MacTech Magazine*, we rely heavily on outside writers for most of the material that appears in our pages. If readers did not participate in the magazine, sending us their ideas and taking the time to write articles, there would be no *MacTech*. *MacTech Magazine* is not a staff of writers sending a constant stream of one-way messages outwards; it's a living, evolving network of readers conversing with one another, educating one another, sharing their knowledge, their experience, their interest, their trials and tribulations and joys and successes in the constantly unfolding story of programming the Macintosh. *MacTech Magazine* doesn't just happen: it's what the community makes it. If we carry reports of future trends and technologies, if we teach useful methods, if we review new books and tools, if we provoke thought, provide help, ride the wave of current interests and concerns, it is only because we reflect the thoughts of our readers, who speak through our pages.

You are invited to involve yourself in this exciting conversation amongst readers. No matter who you are, no matter what your credentials may be, if you have a tale to tell, a trick to share, a technique to teach, we want you to consider joining the family of those who write for *MacTech*.

Don't just wait for a topic to be covered or a technique explained in *MacTech!* Take responsibility! Write us an article yourself!

To write for *MacTech*, just send for our Writer's Kit. It's a Microsoft Word file containing the Styles you need to use, and giving lots of helpful advice and information, including all the legal stuff. You can let us know what you're writing about, or, if you want to, you can just write the article and spring it on us when it's done. [**Note:** We also have a need for people willing to make themselves available to write occasional product/book reviews.] If we publish your article, you'll be paid for it!

Write to us, the editorial staff, at editorial@mactech.com (or one of the other addresses listed on page 2 of the magazine). Take the future of *MacTech Magazine* into your own hands!

For Macintosh Programmers & Developers
MacTech MAGAZINE

THE CLASSIFIEDS

SERVICES

Software Engineers

Industry-leading educational software company needs experienced Software Programmers in C/C++ programming for Windows or Macintosh. Excellent salary and benefits package. A great place to live and develop products that truly make a difference. An opportunity without comparison! Send resumé in confidence to:

#145-MAC 2
Debra Franz
Advantage Learning Systems
P.O. Box 8036
Wisconsin Rapids, WI 54495
Fax (715) 424-3414
E.O.E.

Great Products Deserve Great Docs!

Documentation should be an integral part of your product, not an afterthought. Manual Labor specializes in end-user documentation for Macintosh® products, including Balloon Help and Apple Guide, HTML or PDF files, and the ever-popular "dead tree edition." And while we're documenting every nook and cranny of your product, we can simultaneously provide extensive usability feedback, resulting in fewer bugs, a simpler user interface, reduced support costs, and better reviews.

Contact us today to tap the expertise our team has already brought to companies like Bare Bones Software, id Software, ResNova Software, Management Software, Quasar Knowledge Systems, and Scantron Quality Computers. And see why we're the best choice to document *your* next Mac product.

Jerry Kindall
810/445-9477
kindall@manual.com
http://www.manual.com/

MANUAL LABOR *We Wrote the Book!*
Macintosh Technical Communication Experts

The Law Office of Bradley M. Sniderman

California Lawyer focusing on Intellectual Property, Corporate, Commercial and Contract law, as well as Wills and Trusts.
If you are looking to protect your software with Copyright or Trademark protection, or if you need help establishing or maintaining your business, please give me a call or an e-mail. Reasonable fees.

(310) 553-4054
Brad@Sniderman.com

Visit MacTech® Magazine's Web site!
http://www.mactech.com

http://www.scientific.com

Professional software developers looking for career opportunities should check out our web site. We offer nationwide employment assistance, resume help, marketability assessment, and never a fee to the applicant. 800-231-5920, Fax 800-757-9003 eMail:das@scientific.com

Scientific Placement, Inc.

PSI Scientific Systems

PERCEPTIVE SCIENTIFIC INSTRUMENTS, INC.

Macintosh Developers for US & UK

Perceptive Scientific Instruments, Inc. is a global leader in the development and marketing of digital imaging systems for genetics applications. The company has offices in the United States and the United Kingdom.

We are currently looking for Macintosh Software Developers for both our US and UK locations to work on our next generation products for the genetics market place. Candidates should have 2-5 years of Macintosh programming experience in C/C++ with strong analytical skills. Experience with object oriented programming methodologies will be a plus.

Please send your resumé to H. Baxi, PSII, 2525 S. Shore Blvd., League City, TX 77573, USA or fax to +1 (281) 538-2222 or email to hbaxi@persci.com.

by Steve Sisak

Here is an easy technique for determining if a point lies within a given graphic object of arbitrary complexity: simply draw the object into an offscreen bitmap, then check if the desired pixel has been changed.

In more detail, the steps are:

- Create an offscreen black-and-white bitmap, one pixel high by 16 pixels wide (actually it only needs to be one pixel wide, but QuickDraw insists that you allocate an even number of bytes).
- Erase this bitmap to white.
- Draw the shape into the bitmap in solid black.
- Test the desired pixel to see if it has been set to black.

Here is some actual code that shows you how to hit test a polygon in this way. First, an auxiliary routine to aid the setup of an offscreen black-and-white GrafPort (since GWorlds are overkill for this purpose):

```
PROCEDURE SetBitsTo
  (
    VAR TheBits : BitMap
  );
  (* sets the portBits of the current grafPort, also
    changing the portRect, clipRgn, visRgn to match
    the bounds of the bitmap. *)

    VAR
        CurPort : GrafPtr;

BEGIN
    SetPortBits(TheBits);
    MovePortTo(0, 0);
    SetOrigin(TheBits.bounds.left, TheBits.bounds.top);
    PortSize
      (
        TheBits.bounds.right - TheBits.bounds.left,
        TheBits.bounds.bottom - TheBits.bounds.top
      );
    GetPort(CurPort);
    ClipRect(CurPort^.portRect);
    CopyRgn(CurPort^.clipRgn, CurPort^.visRgn)
END SetBitsTo;
```

Then, the actual routine that hit tests a point against a polygon:

```
PROCEDURE PtInPoly
  (
    pt : Point;
    poly : PolyHandle
  ) : BOOLEAN;
  (* is the specified point within the given polygon. *)

    VAR
        PreviousPort : GrafPtr;
        TempPort : GrafPort;
        TempBits : BitMap;
        TempBitsWord : ShortCard; (* a 1-by-16 offscreen
                                     bitmap *)

BEGIN
    GetPort(PreviousPort);
    OpenPort(ADR(TempPort));
    TempBits.bounds.top := pt.v;
    TempBits.bounds.bottom := pt.v + 1;
    TempBits.bounds.left := pt.h - 15;
        (* so desired pixel is in bit 0 *)
    TempBits.bounds.right := pt.h + 1;
    TempBits.rowBytes := 2;
    TempBits.baseAddr := ADR(TempBitsWord);
    TempBitsWord := 0; (* set all pixels to white *)
    SetBitsTo(TempBits);
    PaintPoly(poly); (* draw polygon in black *)
    ClosePort(ADR(TempPort));
    SetPort(PreviousPort);
    RETURN
        ODD(TempBitsWord)
END PtInPoly;
```

It shouldn't be hard to see how to adapt the basic idea to other sorts of graphic objects, or even to other graphics architectures besides QuickDraw.

Lawrence D'Oliveiro
<ldo@geek-central.gen.nz>

Want to share a tip with the community and get paid for it? Send it in to <mailto:tips@mactech.com>

Send us your tips or we'll install EvenBetterBusError on your machine! On the other hand, we might just pay you ***$25*** *for each tip we use, or* ***$50*** *for Tip of the Month. You can take your award in goods, subscriptions or US$. Make sure any code compiles, and send tips (and where to mail your winnings) to our* ***Tips e-mail address*** *at tips@mactech.com. (See page 2 for our other addresses.)*

by Nicholas C. "nick.c" DeMello <online@mactech.com>

Shareware: Software In a Bottle

There are three great myths in modern society. One concerns a man named Elvis, another involves a place known as Roswell, and the third is the idea that you can't make money writing shareware. I can't say much about the first two, but shareware is a powerful and effective marketing method. It isn't easy to make a living building shareware products, but many folks do. This month we're going to tour some online resources that you should explore if you intend to make money through shareware.

The Voice of Experience

One of the first games I remember playing on the Macintosh was called Scarab of RA. It's not surprising that Rick Holzgrafe, the author of that classic shareware product, has something to say about making a great shareware product. Rick's shareware company (Semicolon Software) hosts eight pages of suggestions for aspiring shareware authors. If you're interested in creating shareware, you should read these pages.

Folks who move files to or from a Mac by FTP know who Peter Lewis is. The author of Anarchie and NetPresenz, Peter makes his living as a full time shareware author and few people have better insight into the shareware industry. Peter and Jeremy Nelson have two pages of practical advice for the new shareware author.

A fundamental and controversial issue of shareware authoring is deciding how to encourage your user into paying a shareware fee. While everyone was arguing their opinion on this issue, shareware author Colin Messitt conducted a case study. Colin released a single copy of a shareware program whose installer had a 50% of chance of creating either of two subtly different versions. One version was completely functional, but periodically reminded the user to register. The other version was hobbled in such a way that the user could explore the programs functionality, but not seriously make use of the program until it was registered. Colin found folks registered the hobbled version 5 times more frequently than they did the unrestricted version. Read the details of this study online at shareware.org.

Rick Holzgrafe's Successful Shareware Pages
<http://www2.Semicolon.com/Rick/ShareSuccess/Shareware1.html>
Peter N Lewis & Jeremy Nelson on Writing Shareware for a Living
<http://www.stairways.com/programming/sharewareauthor.html>
<http://www.stairways.com/programming/sharewareauthor2.html>
"Why do people register shareware?" by Colin Messitt
<http://www.shareware.org/gazer7/why.htm>

Support for the Shareware Author

Possibly the most impressive track record in shareware development belongs to Andrew Welch. For three years Andrew published shareware products under the aegis "Ambrosia." In November of 1992 he released the game Maelstrom, which won MacUser's 1993 Best Shareware Game award. Then, in August of 1993 Andrew founded Ambrosia Software, Inc. which has become one of the most successful Mac OS shareware companies to date. Today, Ambrosia not only offers a variety of award winning games (Chiral, Apeiron, and many more), but they offer the possibility of partnering their considerable experience and resources with new shareware authors and artists. Check out the Ambrosia "TechWorks" pages for the details.

Kagi is a payment processing service, that allows any shareware author to enjoy the luxury of having their own registration department. For a modest percentage of your shareware fees, Kagi will supply you with a registration program (written by Peter Lewis) for distribution with your product. The program allows users to register your shareware electronically, through the mail, or via a web site (payment by US check, money order, Visa, Mastercard, American Express, or cash). Kagi will handle all the registration issues then send you a single check and a detailed registration report each month. If you are at all serious about shareware, consider Kagi.

A number of shareware associations offer support to shareware authors. The most prominent of these, the Association of Shareware Professionals, also hosts a short FAQ about shareware distribution. If you do nothing else, read the ASP's answer to the question "What is shareware?"

Ambrosia's Invitation to Shareware Authors
<http://www.ambrosiaSW.com/TechWorks/Dev.html>
Kagi
<http://www.kagi.com/>
The Association of Shareware Professionals
<http://www.asp-shareware.org/>
The Shareware FAQ, by Mitchell Friedman
<http://www.asp-shareware.org/sharewar.html>
The Elite Shareware Author's Group
<http://www.edepot.com/esagindex.html>
The Shareware Authors Network
<http://www.bsoftware.com/snetwork.htm>

These and other links are available from the MacTech Online web pages <http://www.mactech.com/online/>.

NEWSBITS

by Jessica Courtney

Vibe Now Available on Mac OS

Visix Software Extends Platform Support for Powerful Java Development Tool

Visix Software Inc. announced the general availability of Vibe Enterprise for Windows 95, Windows NT x86, Windows NT Alpha, Solaris, Linux, AIX, HP-UX, IRIX, Macintosh and OS/2. Visix also announced the general availability of Vibe DE for Macintosh, OS/2, Windows NT Alpha, HP-UX and IRIX. Vibe DE is already available for development on Windows 95, Windows NT x86, Linux, AIX and Solaris.

Vibe Enterprise enables Java developers to build powerful database applications. Vibe Enterprise leverages Vibe DE, an Integrated Development Environment (IDE) with a collection of powerful classes used to build and deploy Java-based business solutions ranging from standalone departmental applications to sophisticated distributed applications.

The Vibe IDE includes an intuitive user interface, a compiler, debugger, editing tools, interface construction tools and extensive runtime classes. It enables developers to do rapid prototyping, immediate testing and development of business-critical Java applications for deployment across multiple desktop and server platforms. Vibe Enterprise adds additional tools and classes for Forms and Fields, database connectivity and a Graphical Query Builder.

Vibe is designed to maximize code reuse by leveraging Visix's mature foundation classes, which also form the basis of Visix's flagship C/C++ product, the Galaxy Application Environment. To gain additional productivity through reuse, Vibe enables organizations to incorporate their investments in standard distributed object technologies through application integration with ActiveX controls.

Vibe DE is available at the introductory price of US$49.95. Vibe Enterprise is available at US$1995.00.
<http://www.visix.com>.

QuickGuard From MicroGuard

Micro Macro Technologies Ltd. (MMT), announces their newest feature to the MicroGuard PC hardware protection key — "QuickGuard". QuickGuard allows anyone to wrap an entire software application within a protective shield in ten minutes or less, even without prior programming or copy-protection experience.

This newest release joins the full range of MicroGuard copy-protection products which include MicroGuard PC and MicroGuard Plus for the Macintosh. With the development of all of their products, MMT focuses on easy-implementation and unimpaired plug-and-forget technology for the end-user.
<http://www.micromacro.com>.

Blue World Technology Acquired by Claris

Blue World Communications, Inc. and Claris Corporation announced that Blue World's Lasso for Web-enabling databases has been integrated into FileMaker Pro 4.0 for Windows 95, Windows NT, Windows 3.1 and Mac OS from Claris Corporation. Under the terms of the agreement, Claris has acquired the Lasso technology and licensed certain rights back to Blue World. Blue World plans to continue enhancing Lasso and distributing new versions of Lasso as a commercial product.

FileMaker Pro 4.0 embeds a subset of Lasso 2.0 technology as FileMaker Pro's web companion. This integration provides FileMaker Pro 4.0 owners with out-of-the-box Web database publishing and server capabilities.

Blue World will continue to sell the full Lasso 2.0 product to users of both FileMaker Pro 3.0 and FileMaker Pro 4.0 for Mac OS. Lasso 2.0 Web-enables FileMaker Pro 3.0 and a Lasso 2.0 Server version adds a stand-alone Web server with integrated Lasso 2.0 CGI.

Lasso 2.0 also provides additional features for FileMaker Pro 4.0 databases beyond what is included in the new version, including an "inline" command to process multiple database operations within a single HTML file; a "log" command to log traffic statistics to target databases and/or text files; and Apple Event control of other server applications.
<http://www.blueworld.com>, <http://www.claris.com>.

New InterLok Version Extends PACE Anti-Piracy Leadership in Software Authorization

PACE Anti-Piracy has introduced Version 1.0.2 of its InterLok Software Authorization System. This update to InterLok for Macintosh and InterLok Pro for Macintosh, PACE's software-only packages for turnkey trialware creation and copy protection, includes improvements to InterLok's anti-hacking mechanisms as well as bug fixes. In addition, this release adds to InterLok Pro the ability to protect data files in association with an application, along with PowerPC-native encryption and decryption routines for enhanced performance for Power Macintosh users.

The new data file protection capability extends the power of InterLok Pro to developers who wish to distribute secure versions of their run-time database solutions. For example, developers of FileMaker Pro solutions can now use InterLok Pro to easily create time-limited trialware versions of their products, and then enable them remotely by means of a machine-unique key, ensuring that they collect revenue when their products are used.
<http://www.paceap.com>.

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

LIST OF PRODUCTS

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

develop

by Tim Monroe, Apple Computer, Inc.

Catching a WAVE

Playing WAVE Files on the MacOS

By far the most common type of sound file on Windows computers (and hence on personal computers in general) is the .WAV format file, also called a WAVE file or a waveform file. Collections of WAVE files are easily accessible on the Internet and elsewhere, and the number of WAVE files available for downloading far outstrips the number of Macintosh sound files. So what's a loyal Macintosh programmer to do when confronted with audio content in WAVE format? The Sound Manager's SndStartFilePlay function won't play WAVE files (not yet, at least), but it's easy to use other Sound Manager capabilities to play the data contained in a WAVE file.

In this article, I'll show how to open, parse, and play a WAVE file. In fact, this is a surprisingly simple thing to do, largely because the digitized sound data in a WAVE file is stored in pretty much the same format as digitized sound data on the Mac. So, all we need to do is extract the sound data from the WAVE file and pass it to standard Sound Manager routines. If you've never used the low-level Sound Manager interfaces, this will be a good introduction. Along the way, you'll also learn how to deal with the WAVE file's chunk architecture and the little-endian byte ordering used on Windows.

If you're not inclined to work with sound files at this low level, don't despair. At the end of this article, I'll show an alternate strategy for playing WAVE files that uses the QuickTime APIs instead of the Sound Manager. Indeed, if you're really averse to low-level coding, you should skip straight to the section "Surfing with QuickTime" and read about the high-level method. Otherwise, strap on your wet suit, and let's go!

The Chunk Architecture

A WAVE file contains digitized (that is, sampled) sound data, just like most sound files and resources on the Macintosh. A WAVE file also contains information about the format of the sound data, such as the number of bits per sample and the number of channels of audio data (mono vs. stereo). The various kinds of data in a WAVE file are isolated from one another using an architecture based on *chunks*. A chunk is simply a block of data together with a chunk header, which specifies both the type of the chunk and the size of the chunk's data. **Figure 1** illustrates the basic structure of a chunk.

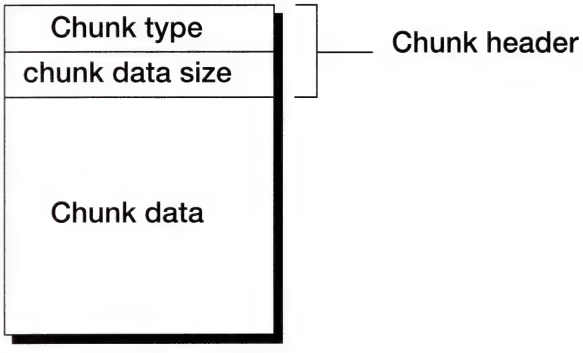


Figure 1. *The basic structure of a chunk.*

A WAVE file always contains at least two chunks, a *data chunk* that contains the sampled sound data, and a *format chunk* that contains information about the format of the sound data. These two chunks (and any others that might occur in the file) are packaged together inside of another chunk, called a *container chunk* or a *RIFF chunk*. Like any chunk, the RIFF chunk has a header (whose chunk type is 'RIFF') and some chunk data. For RIFF chunks, the chunk data begins with a data format specifier followed by all the other chunks in the file. The format specifier for a WAVE file is 'WAVE'. **Figure 2** shows the general structure of any WAVE file.

Tim Monroe is a software engineer on Apple's QuickTime VR team. In his first eight years at Apple, he worked on the *Inside Macintosh* team, where he wrote developer documentation for QuickDraw 3D, QuickTime VR, the sound and speech technologies, and a host of other APIs. You can contact him at <monroe@apple.com>.

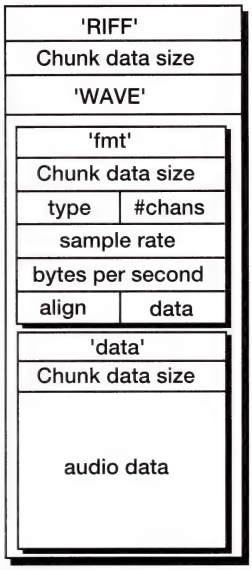


Figure 2. *The basic structure of a WAVE file.*

The first thing we'll do is define some data structures that will help us extract the information from a chunk header or from the chunk data. In C, we can represent a chunk header like this:

```
typedef struct ChunkHeader {
  FOURCC    fChunkType;      // the type of this chunk
  DWORD     fChunkSize;      // the size (in bytes) of the chunk data
} ChunkHeader, *ChunkHeaderPtr;
```

Here, we're using standard Windows data types; on the Macintosh, these types are #define'd to more familiar types:

```
#define WORD    UInt16
#define DWORD   UInt32
#define FOURCC  OSType
```

We can represent a format chunk like this:

```
typedef struct FormatChunk {
  ChunkHeaderfChunkHeader;    // the chunk header; fChunkType == 'fmt '
  WORD    fFormatType;        // format type
  WORD    fNumChannels;       // number of channels
  DWORD   fSamplesPerSec;     // sample rate
  DWORD   fAvgBytesPerSec;    // byte rate (for buffer estimation)
  WORD    fBlockAlign;        // data block size
  WORD    fAdditionalData;    // additional data
} FormatChunk, *FormatChunkPtr;
```

And, finally, we can represent the relevant parts of a RIFF chunk like this:

```
typedef struct RIFFChunk {
  ChunkHeaderfChunkHeader;    // the chunk header; fChunkType == 'RIFF'
  FOURCC    fFormType;        // the data format; 'WAVE' for waveform files
                              // additional chunk data follows here
} RIFFChunk, *RIFFChunkPtr;
```

It's very easy to parse a file that is structured into chunks. You simply begin at the start of the file data, which is guaranteed to be the beginning of the container chunk. You can find the first subchunk by skipping over the container chunk header and any additional container chunk data. And you can find any succeeding subchunks by skipping over the subchunk header and the number of bytes occupied by the subchunk data. Listing 1 shows how to find the beginning of a chunk of a specific type.

Listing 1: Finding a chunk of a specific type

```
OSErr GetChunkType (short theFile, OSType theType,
                    long theStartPos, long *theChunkPos)
{
  OSErr         myErr = noErr;
  long          myLength = sizeof(ChunkHeader);
  UInt32        myOffset;
  ChunkHeader   myHeader;
  Boolean       isFound = false;
  // set file mark relative to start of file
  myErr = SetFPos(theFile, fsFromStart, theStartPos);
  if (myErr != noErr)
    return(myErr);
    // search the file for the specified chunk type
  while (!isFound && (myErr == noErr)) {
    // load the chunk header
    myErr = FSRead(theFile, &myLength, &myHeader);
    if (myErr == noErr) {
      if (myHeader.fChunkType == theType) {
        isFound = true;         // we found the desired chunk type
        myErr = GetFPos(theFile, theChunkPos);
        *theChunkPos -= myLength;
      } else {
        if (myHeader.fChunkType == kChunkType_RIFF)
          myOffset = sizeof(FOURCC);
        else
          myOffset = Swap_32(myHeader.fChunkSize);
        if (myOffset % 2 == 1)  // make sure chunk size is even
          myOffset++;
          myErr = SetFPos(theFile, fsFromMark, myOffset);
      }
    }
  }
  return(myErr);
}
```

The function GetChunkType starts searching the file data at a location (theStartPos) passed to it, which is assumed to be the start of a chunk. GetChunkType reads in the chunk header and looks to see if it has found the chunk of the desired type. If so, it returns the file position of the first byte of the chunk header. Otherwise, GetChunkType figures out where in the file the next chunk begins. If the current chunk is a container chunk, then the next chunk begins after the data format specifier; otherwise, the next chunk is to be found after the current chunk's data, whose size is specified in the chunk header. (Notice that in determining the size of the current chunk, we need to use the macro Swap_32; see "Which End Is Which?" for an explanation of why this is necessary.)

Once we know how to find the beginning of a particular chunk, it's easy to get the chunk's data. Listing 2 defines a function GetChunkData that returns a pointer to a buffer of memory containing both the chunk header and the data in the chunk.

Listing 2: Getting a chunk's data

```
ChunkHeaderPtr GetChunkData
        (short theFile, OSType theType, long theStartPos)
{
  long          myFPos;
  ChunkHeader   myHeader;
  Ptr           myDataPtr = NULL;
  OSErr         myErr = noErr;
  long          myLength;
  // get position of desired chunk type in file
  myErr = GetChunkType(theFile, theType,
                       theStartPos, &myFPos);
  // set file mark at the start of the chunk
  if (myErr == noErr)
    myErr = SetFPos(theFile, fsFromStart, myFPos);
if (myErr == noErr) {
    myLength = sizeof(myHeader);
    // load the chunk header
    myErr = FSRead(theFile, &myLength, &myHeader);
    if (myErr != noErr)
      return(NULL);
    // set file mark at the start of the chunk header
    myLength += Swap_32(myHeader.fChunkSize);
```

```
    myErr = SetFPos(theFile, fsFromStart, myFPos);
  }
  if (myErr == noErr) {
    myDataPtr = NewPtrClear(myLen);
    myErr = MemError();
    if (myDataPtr != NULL)
      myErr = FSRead(theFile, &myLength, myDataPtr);
  }
  if (myErr != noErr) {
    DisposePtr(myDataPtr);
    myDataPtr = NULL;
  }
  return((ChunkHeaderPtr)myDataPtr);
```

GetChunkData calls GetChunkType to find the beginning of the chunk of the desired kind; then it reads the chunk header to find the size of the chunk data. (Once again, we've used the macro Swap_32 to massage the chunk data length as it's stored in the file.) Finally, GetChunkData allocates a buffer large enough to hold the entire chunk (header and data) and returns the pointer to the caller.

Here's a pleasant surprise: the functions GetChunkType and GetChunkData are simply slightly modified C language translations of the functions MyFindChunk and MyGetChunkData found in *Inside Macintosh: Sound* (pages 2–63 and 2–65, respectively). We could use those functions, suitably modified, because the AIFF format (defined by Apple and described in *Inside Macintosh: Sound*) and the RIFF format (defined by Microsoft and used for WAVE files) are both chunk-based formats, which descend from a common parent. See the sidebar "A Brief History of IFF" for details.

Which End Is Which?

Now we know how to find a chunk in a RIFF file and read the data in that chunk into memory. We've already seen, however, that we sometimes need to play with that data before we can use it. That's because of a difference in the way multi-byte data is accessed on Motorola and Intel processors. Motorola processors in the 680x0 family expect multi-byte data to be stored with the most significant byte lowest in memory. This is known as "big-endian" byte ordering (because the "big" end of the data value comes first in memory). Intel processors used for Windows machines expect multi-byte data to be stored with the least significant byte lowest in memory; this is known as "little-endian" byte ordering. (See Figure 3, which shows the value 0x12345678 stored in both Motorola 680x0 and Intel formats.) The PowerPC family of processors supports both big- and little-endian byte orderings, but uses big-endian when running the MacOS.

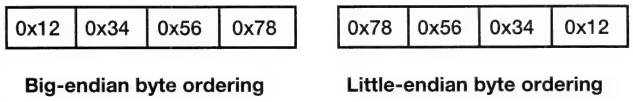


Figure 3. *Big- and little-endian byte ordering.*

The data stored in a WAVE file is little-endian. As a result, to use that data in a Macintosh application, we need to convert the little-endian data to big-endian data — but only for data that is larger than 8 bits. For instance, the chunk data size field in a chunk header is 4 bytes long, so we need to swap the bytes using our macro Swap_32. Later, we'll also need to swap the two bytes in a 16-bit field, so we can define these macros:

```
#define Swap_32(value)          \
    (((((UInt32)value)<<24) & 0xFF000000)  | \
     ((((UInt32)value)<< 8) & 0x00FF0000)  | \
     ((((UInt32)value)>> 8) & 0x0000FF00)  | \
     ((((UInt32)value)>>24) & 0x000000FF))

#define Swap_16(value)          \
    (((((UInt16)value)>> 8) & 0x000000FF)  | \
     ((((UInt16)value)<< 8) & 0x0000FF00))
```

You might be wondering why we didn't need to swap bytes when reading the chunk type from a file. That's because a chunk type is a sequence of four (8-bit) characters. When reading individual characters, no byte swapping is necessary. A byte in little-endian byte ordering is the same as a byte in big-endian byte ordering. For this same reason, we don't need to do any work when reading 8-bit audio samples from the WAVE file and (eventually) passing them to the Sound Manager.

For 16-bit audio samples, however, the bytes *do* need to be swapped before they can be passed to the Sound Manager. You could do this yourself, by loading all the data into a buffer and then running through the buffer swapping every pair of bytes. Better yet, if you're using Sound Manager version 3.1 or later, you can have the Sound Manager do the byte swapping for you. You do this by invoking the 'sowt' data decompressor on the (uncompressed) 16-bit audio data. (Notice that 'sowt' is 'twos' with the bytes swapped; 16-bit data is stored in a two's-complement format.) See Listing 6 for the exact details of invoking this codec on 16-bit audio data.

It's worth mentioning that RIFF has a counterpart, RIFX, that uses Motorola byte ordering. A RIFX file is exactly like a RIFF file except that the container chunk has the ID 'RIFX' and all multi-byte values are stored in big-endian format. Naturally, if you encounter a RIFX file, you can dispense with all the byte swapping.

Opening the WAVE File

Of course, before we can start reading the data in a WAVE file, we need to open the file. On the Macintosh Operating System, a WAVE file is contained entirely in a file's data fork. Listing 3 defines a simple function that lets the user select a WAVE file for playing and then calls FSpOpenDF to open the file for reading.

Listing 3: Opening a WAVE file

```
short OpenWaveFile (void)
{
  StandardFileReply myReply;
  SFTypeList        myTypeList = {'WAVE', 'BINA', 0, 0};
  short             myRefNum;
  OSErr             myErr = noErr;
  // elicit a file from the user
  StandardGetFile(NULL, 2, myTypeList, &myReply);
  if (!myReply.sfGood)
    return(-1);
  // open the file's data fork for reading
  myErr = FSpOpenDF(&myReply.sfFile, fsRdPerm, &myRefNum);
  if (myErr != noErr)
    return((short)myErr);
  else
    return(myRefNum);
}
```

Notice that we're allowing the user to select files whose type is either 'WAVE' or 'BINA'. I've found that files downloaded from the Internet usually have a file type of 'WAVE', while files copied over a local area network from a PC sometimes have the file type 'BINA'. To make sure that we've gotten an actual WAVE file, we can execute this code:

Listing 4: Verifying a WAVE file

```
myDataPtr = GetChunkData(myRefNum, kChunkType_RIFF, 0);
if (myDataPtr != NULL) {
  RIFFChunkPtr myRIFFPtr = (RIFFChunkPtr)myDataPtr;
  FOURCC                                    myFormType;
    myFormType = myRIFFPtr->fFormType;
  DisposePtr((Ptr)myDataPtr);
  if (myFormType != kRIFFType_WAVE)
    return(badFileFormat);
}
```

Using the Sound Manager

Once we've opened a bona fide WAVE file, we can get the sampled sound data from it by calling GetChunkData, like this:

```
ChunkHeaderPtr  myDataPtr = GetChunkData(myRefNum,
kChunkType_Data, 0);
```

If this call succeeds, it returns a pointer to a buffer that contains the entire sound data chunk, from which we can get the actual sound data by skipping over the chunk header. The natural thing to do is play the sound using the Sound Manager's **bufferCmd** sound command. The only relevant parameter to **bufferCmd** is the address of a sound header structure, which tells the Sound Manager where the audio data is and what its properties are. So, we need to get the information about the WAVE sound and put that information into the sound header structure. Listing 5 shows how to read the sound format information from the WAVE file.

Listing 5: Getting the sound format information

```
myDataPtr = GetChunkData(myRefNum, kChunkType_Format, 0);
if (myDataPtr != NULL) {
  FormatChunkPtr  myFormatPtr = (FormatChunkPtr)myDataPtr;
  myFormat = Swap_16(myFormatPtr->fFormatType);
  myNumChannels = Swap_16(myFormatPtr->fNumChannels);
  mySampleRate =
            Long2Fix(Swap_32(myFormatPtr->fSamplesPerSec));
  myBitsPerSample = Swap_16(myFormatPtr->fAdditionalData);
  DisposePtr((Ptr)myDataPtr);
  // currently, we support only standard PCM encoding
  if (myFormat != WAVE_FORMAT_PCM)
    return(badFileFormat);
}
```

As before, we need to swap the bytes on any data read from the file that's longer than 8 bits. The **fFormatType** field of a format chunk specifies the type of WAVE file. Here we support only uncompressed files, which have the type **WAVE_FORMAT_PCM**.

Now we need to fill in a sound header with this data. The Sound Manager defines three different sound headers. Which sound header we use depends on the features of the sound to be played. Because we want to invoke the '**sowt**' decompressor for 16-bit data, we'll use the sound header of type **CmpSoundHeader**. Listing 6 shows how to fill in the sound header.

Listing 6: Filling in a sound header

```
CmpSoundHeader  mySoundHeader;

// fill in a compressed sound header structure,
mySoundHeader.samplePtr =               // skip the chunk header
                    (Ptr)myDataPtr + sizeof(ChunkHeader);
mySoundHeader.numChannels = myNumChannels;
mySoundHeader.sampleRate = mySampleRate;
mySoundHeader.loopStart = 0;
mySoundHeader.loopEnd = 0;
mySoundHeader.encode = cmpSH;
mySoundHeader.baseFrequency = kMiddleC;
mySoundHeader.numFrames =
                (Swap_32(myDataPtr->fChunkSize) * 8) /
                (myNumChannels * myBitsPerSample);
mySoundHeader.markerChunk = NULL;
mySoundHeader.format = kCompressType_None;
mySoundHeader.futureUse2 = 0;
mySoundHeader.stateVars = NULL;
mySoundHeader.leftOverSamples = NULL;
mySoundHeader.compressionID = notCompressed;
mySoundHeader.packetSize = 0;
mySoundHeader.snthID = 0;
mySoundHeader.sampleSize = myBitsPerSample;
// remember that data in a WAVE file is stored in little-endian byte ordering;
// accordingly, for 16-bit sounds, we need to invoke the 'sowt' decompressor
// that will swap bytes for us (available only in Sound Manager 3.1 and later)
if (myBitsPerSample == 16) {
  mySoundHeader.format = 'sowt';
  mySoundHeader.compressionID = fixedCompression;
}
```

This is all straightforward. Note how easy it is to invoke the 'sowt' decompressor; we just define the compression type and compression ID to the appropriate values. The next step is to send a sound command to a sound channel. We construct a sound command like this:

```
SndCommand      mySoundCommand;
// now play the sound using a bufferCmd
mySoundCommand.cmd = bufferCmd;
mySoundCommand.param1 = 0;          // unused with bufferCmd
mySoundCommand.param2 = (long)&mySoundHeader;
```

Finally, Listing 7 shows how to allocate a sound channel and pass the sound command to that channel by calling **SndDoImmediate**:

Listing 7: Playing a buffer of sound data

```
SndChannelPtr   mySoundChannel = NULL;
// allocate a sound channel
myErr = SndNewChannel(&mySoundChannel, sampledSynth,
                                                initMono, NULL);
if (mySoundChannel != NULL)
  myErr = SndDoImmediate(mySoundChannel, &mySoundCommand);
```

At this point, if everything has gone according to plan, the Sound Manager will begin playing the sound that we've loaded from the WAVE file.

Looking for the Really Big Wave

Here I've shown how to open, parse, and play both 8-bit and 16-bit mono and stereo WAVE files. For commercial products, however, you'll probably want to make a few enhancements to this code. For instance, I've supposed that the sound data can always fit into the available memory. For very large WAVE files, this might not be true. In that case, you can implement a double-buffering scheme by reading small portions of the sampled sound data into one of two (or more) buffers and then playing that data using the **bufferCmd** sound command. This method for handling large sound files is suggested by Olson, 1995. Note that the use of the **SndPlayDoubleBuffer** function (as illustrated in Day, 1991) is no longer recommended.

In addition, a more complete WAVE-playing application would want to add support for the standard compression algorithms you're likely to find in compressed WAVE files, as well as beef up the sound management capabilities so that you can adjust the volume and balance of the sound, change its pitch, and so forth. Since you're using the Sound Manager to play the WAVE files, you have access to the full range of its capabilities. For instance, you can install callback routines to trigger the deallocation of the sound buffers when the sounds are finished playing. All of these capabilities are illustrated in a code sample called SndPlayDoubleBuffer written by Mark Cookson of Apple DTS that's included on the Developer CD series. (Look in the folder Sound inside the Snippets folder; better yet, check out the URL listed at the end of this article.)

Surfing with QuickTime

QuickTime, as I've reported elsewhere (Monroe, 1997), has evolved into a cross-platform vehicle for the authoring, delivery, and playback of digital multimedia content. One thing this means is that QuickTime is very, very good at playing sounds. Indeed, current versions of QuickTime are able to handle 8- and 16-bit uncompressed WAVE files, and the forthcoming version 3.0 (on both MacOS and Windows) will handle compressed WAVE data as well. So, you can use the standard QuickTime APIs to open and play WAVE files, unless you absolutely need to use Sound Manager capabilities that are not directly supported by QuickTime.

The basic idea to using QuickTime to play WAVE files is to call the NewMovieFromFile function. If the specified file does not contain a movie resource, NewMovieFromFile tries to locate a *movie data import component* that converts the file data into a movie, which can then be played using standard QuickTime functions. Listing 8 shows a routine that opens and plays a waveform file using the QuickTime API.

Listing 8: Playing WAVE files with QuickTime

```
void PlayWaveUsingQuickTime (void)
{
  StandardFileReply   myReply;
  SFTypeList          myTypeList = {'WAVE', 0, 0, 0};
  short               myRefNum;
  Movie               myMovie;
    // elicit a file from the user
  StandardGetFile(NULL, 2, myTypeList, &myReply);
  if (!myReply.sfGood)
    return;
    // use QuickTime routines to play the sound
  EnterMovies();
  OpenMovieFile(&myReply.sfFile, &myRefNum, fsRdPerm);
  NewMovieFromFile(&myMovie, myRefNum, NULL, NULL, 0, NULL);
  if (myRefNum != 0)
    CloseMovieFile(myRefNum);
  StartMovie(myMovie);
  while (!IsMovieDone(myMovie))
    MoviesTask(myMovie, 0);
  DisposeMovie(myMovie);
}
```

PlayWaveUsingQuickTime begins by calling StandardGetFile to elicit a file from the user. Then it opens the file and calls NewMovieFromFile to create a movie from the sampled sound data in the file. The remainder of the function simply starts the movie playing and waits until it's finished.

Now I suspect you're wondering why we bothered at all with parsing chunks and swapping bytes, if QuickTime can do it all for us? That's a good question. First, your application might be concerned primarily with playing sounds, and you might already have developed an architecture for tracking and disposing of buffers of sound data. In that case, the techniques shown earlier in this article are likely to integrate into your existing code more easily than the QuickTime technique shown in Listing 8. Also, as noted earlier, using the Sound Manager directly gives you access to a large set of tools that you can use to modify the sounds being played. QuickTime supplies some of these capabilities, but not all of them. For instance, you can use the Sound Manager's rateCmd to alter the sample rate of a sound already being played. To my knowledge, there is no way to do this using just the QuickTime APIs. Nevertheless, for the vast majority of cases, where you simply want to play a WAVE file and perhaps pause it at various times, the QuickTime solution is simpler and far more elegant.

A Brief History of IFF

The chunk architecture was developed in the mid 1980's by Electronic Arts, an interactive entertainment software developer, in conjunction with Commodore-Amiga. The goal was to be able to store data (particularly multimedia data such as sounds, images, and animation controls) in a format that makes the data easy to move from one operating system to another. A chunk is just a block of data that has both a type and a length. (The representation of a chunk type as a four character sequence was borrowed directly from the Macintosh's use of four character file types, resource types, and so forth.) Electronic Arts defined the structure of chunks and the means of storing chunks in files. This simple structure was designed to make these files easy to parse and create. See Morrison, 1985 for a description of the Interchange File Format (IFF) standard.

Electronic Arts' IFF standard was soon used by Apple as the basis for the Audio Interchange File Format (AIFF) and the Audio Interchange File Format for Compression (AIFF-C) specifications. As the names suggest, Apple used these formats primarily to store audio data such as sampled sound data or MIDI data, along with associated information about that data. In System 7.0, Apple introduced an enhanced Sound Manager that provides system software support for reading and writing AIFF files. Other manufacturers, SGI for example, have also adopted AIFF as a standard sound file format.

In Windows 3.1, Microsoft introduced its own version of the IFF standard: the Resource Interchange File Format (RIFF). RIFF supports a wide range of data types, including bitmaps, color palettes, audio-video interlaced (AVI) data, MIDI data, and waveform data. The Win32 programming interfaces include support for reading and writing RIFF files.

Bibliography and References

- Day, Neil. "Around and Around: Multi-Buffering Sounds". *develop*, The Apple Technical Journal, issue 11 (August 1992), pp. 38–58.
- *Inside Macintosh: QuickTime*, by Apple Computer, Inc. (Addison-Wesley, 1993).
- *Inside Macintosh: QuickTime Components*, by Apple Computer, Inc. (Addison-Wesley, 1993).
- *Inside Macintosh: Sound*, by Apple Computer, Inc. (Addison-Wesley, 1994).
- Monroe, Tim. "The QuickTime Media Layer". *MacTech Magazine*, volume 13, no. 7 (July 1997), pp. 51–54.
- Morrison, Jerry. EA IFF 85 Standard for Interchange Format Files (Electronic Arts, 1985).
- Olson, Kip. "Sound Secrets". *develop*, The Apple Technical Journal, issue 24 (December 1995), pp. 45–55.

URLs

You can find the IFF specification (Morrison, 1985) at many sites on the World Wide Web; try <http://www.sprog.auc.dk/~motr96/sirius/neuro/dev/extrefs/iff_spec.txt>. For the RIFF specification, look at <http://www.seanet.com/HTML/Users/matts/riffmci/riffmci.htm>. For the sample double buffering application SndPlayDoubleBuffer by Mark Cookson, see <http://devworld.apple.com/techsupport/source/SSound.html>.

Acknowledgements

Thanks to Mark Cookson, Peter Hoddie, and Jim Reekes for reviewing this article. MT

by Ingrid Kelly, Apple Computer, Inc.

Introducing the LaserWriter Driver Version 8.5.1

This Article outlines some of the changes and new features that Apple has made with the LaserWriter version 8.5.1 release. It describes new utilities, expanded technology support, and what this means to you, the developer.

Note: This article is a generic overview of the changes in the LaserWriter version 8.5.1 release. There are several more TechNotes on Apple's TechNote web site that provide more detail on how to implement and support some of the new features with your driver or application.

Desktop Printer Utility

In the current LaserWriter 8 driver, the Chooser lists printers which are available on the AppleTalk network (either LocalTalk or AppleTalk) with an NBP type of "LaserWriter." When the user selects one of these printers, LaserWriter 8 creates a desktop printer by sending an Apple Event to the Finder. In the past, this mechanism of setting up printers has served the needs of most users. However, with LaserWriter 8.5.1, Apple is responding to several requests for more flexible desktop printing functionality. To this end, Apple has introduced a new application called Desktop Printer Utility which enables users to create additional types of desktop printers. Users can now create desktop printers which use the Unix lpr protocol for printing, in addition to the regular AppleTalk PAP printers. There are also "hold" printers which represent local print queues and "virtual" printers which represent printers which are not available on the network.

Note: The Desktop Printer Utility will not work on Mac OS 8.0. Due to the parallel development schedules and Finder dependencies, this utility will only work on Mac OS 7.5 through 7.6.1 and Mac OS 8.1 and above.

Types of Desktop Printers Currently Implemented with the Desktop Printer Utility

Virtual- Printer (No Printer Connection):

Many users have asked Apple to provide a mechanism to create desktop printers that save their PostScript output to a file without having to be connected to a network. This is a useful feature for those users who publish or archive PostScript documents as well as for those users preparing to take a document to a service bureau. This can be done with Desktop Printer Utility by having the user set up a 'virtual printer' that saves PostScript files to disk.

Hold- Translator (PostScript):

A type of desktop printer useful for PowerBook users is a "hold" desktop printer. Such a desktop printer is always in hold mode and, as the user prints, the spool files appear in the desktop printer queue but are never printed. When the PowerBook is connected to a network with printers, the user can then copy the spool files from the hold desktop printer queue to a desktop printer that represents a real printer on the network.

TCP/IP/LPR- Printer (LPR):

Many users, especially network administrators at universities, have asked for LPR support for LaserWriter 8. With Desktop Printer Utility, printers that represent TCP/IP printers are supported. When creating a TCP/IP desktop printer, the user provides either the name of the printer or its IP address. When LaserWriter 8.5.1 prints to one of these DTPs, it will send the job using the LPR protocol.

Ingrid Kelly works in Developer Technical Support (DTS) at Apple. Having lived on the East coast for a few years, Ingrid is happy to be back in California where she doesn't have to dig her car out in the wintertime, nor does she have to trudge to school in the snow... uphill both ways! At least she'll have a story to tell her children though!

DTP Utility Customization

Along with the expanded capabilities, Desktop Printer Utility is also customizable. LaserWriter 8.5.1 and Desktop Printer Utility together) support the creation and use of desktop printers known as Custom DTPs. When printed to, Custom DTPs cause the LaserWriter 8.5.1 driver to create a PostScript file and to launch an application that can post-process the PostScript. This post-processing application can do anything it likes with the PostScript file, such as converting the PostScript into another file format, transferring the file to another location using a modem or a network connection, or displaying the PostScript file to the user. Customization simply requires a developer to create a few resources and the LaserWriter 8.5.1 driver will do the rest. Please see "Customizing Desktop Printer Utility" in this issue for more information.

The Extended 'PAPA'

In order to support the new types of desktop printers for Desktop Printer Utility, Apple needed to grow the 'PAPA' resource stored in the driver. It has grown from 103 bytes to 1024 bytes. The new 'PAPA' looks just like an old style 'PAPA'; that is, the 'PAPA' resource begins with three packed Pascal strings representing the printer name, the network type (usually LaserWriter), and the printer's zone. After these strings comes the 4-byte network address. The new DTP data begins at the 104th byte in the resource. The data from byte 104 to 1024 (1-based indices) is a series of tagged values where each value begins with a 4-byte tag, a 2-byte length value, and then the actual data. For more information on the extended 'PAPA' resource, please see Apple's TechNote web site.

Redesigned Custom Page Size Support

With LaserWriter 8.4.x, Apple temporarily removed its custom page size support. However, with LaserWriter 8.5.1, this functionality has been redesigned and returns with a much improved user interface.

When a printer's PPD file has a "*CustomPageSize True:" entry in it, then LaserWriter 8.5.1 displays a "Custom Page Sizes" panel in the Page Setup dialog.

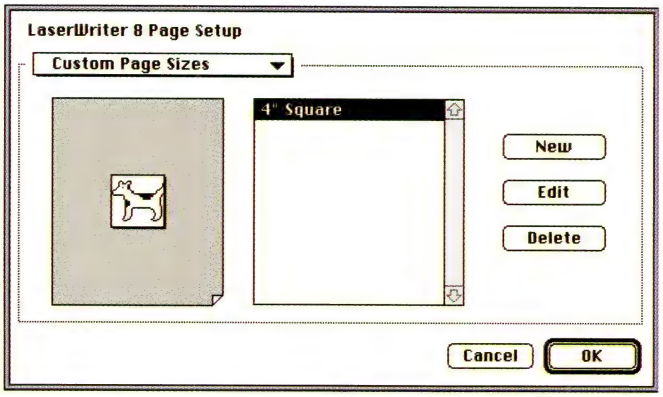


Figure 1.

Each new page size has a width and height as well as margins on the four edges. In addition to these standard paper descriptors, each page has a width and height offset for the sheet it is being printed upon. This new panel allows the user to create and modify named page sizes. You should consult the PPD 4.3 specification from Adobe for more details on PPD files and the parameters for custom page sizes.

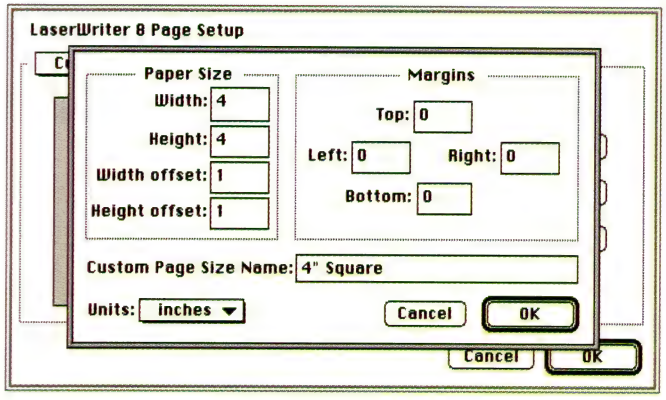


Figure 2.

Notes and Limitations

- Page sizes that are created are shared among all printers that support custom page sizes; the list of custom page sizes is sticky to the driver, not to the PPD.
- If a print record specifies a custom page size, but the current printer does not support custom page sizes, then the paper pop-up in the 'Page Attributes' panel will specify 'other' as the current paper size.
- The standard print record places some limitations on the number and sizes of custom page sizes that a user may create. For a given paper size there can only be two custom paper size entries and they must have different imageable areas. If an application uses the extended print record, this restriction no longer applies.

PostScript Level 3- CopyBits/StdPix expanded support

LaserWriter 8.5.1 supports the PostScript Level 3 deep masked image extension. This extension allows Apple to support the QuickDraw transparent CopyBits() mode as well as more fully support the maskRgn parameter.

The support for transparent and clipped images has been placed in LaserWriter 8.5.1's implementation of QuickDraw's CopyBits() call. In particular, LaserWriter 8.5.1 recognizes the maskRgn CopyBits() parameter and honors it (Inside Macintosh: Imaging With QuickDraw, p.3-112). LaserWriter 8.5.1 also respects the transparent transfer mode when passed in the 'mode' parameter for CopyBits() (Inside Macintosh: Imaging With QuickDraw, p. 4-39).

For an in-depth look at LaserWriter 8.5.1's support for

CopyBits(), see Apple's TechNote web site.

For more information on PostScript Level 3, please contact Adobe Systems Incorporated.

PrGeneral opcodes

Table 1 lists the current PrGeneral opcodes that are implemented/supported by the LaserWriter 8.5.1 driver. With the LaserWriter 8.5.1 release, the driver added complete support for the enableColorMatchingOp opcode for ColorSync. See the ColorSync 2.1.x Support section for further details.

Opcode	Operation
4	getRslDataOp
5	setRslOp
6	draftBitsOp
7	noDraftBitsOp
8	getRotnOp
9	NoGrayScl (not used by LaserWriter 8)
10	getPSInfoOp
11	PSIntentionsOp
12	enableColorMatchingOp (LaserWriter 8.5.1 and above)
13	registerProfileOp (ColorSync 1.x only; not used by LaserWriter 8)
14	PSAdobeOp
15	PSPrimaryPPDOp
16	kLoadCommProcsOp
17	kUnloadCommProcsOp
18	kExtendPrintRecOp (LaserWriter version 8.4 and above)
19	kGetExtendedPrintRecOp (LaserWriter version 8.4 and above)
20	kPrinterDirectOpCode (not used by any LaserWriter driver)
21	kSetExtendedPrintRecOp (LaserWriter version 8.4 and above)
22.	kPrVersionOp (LaserWriter version 8.4 and above).

Table 1.

Please refer to "Meet PrGeneral, the Trap that Makes the Most of the Printing Manager" in develop Issue 3 for more information on PrGeneral.

ColorSync 2.1.x Support

New Tag Support- 'dsc2'

ColorSync 2.1.x provides a new profile tag that contains the profile's name in several languages. This tag, 'dsc2,' provides localized names for a given profile. LaserWriter 8.5.1 looks inside of a profile for this tag and tries to match the language of the system font against the list of languages in the 'dsc2' tag. If a match is found, the localized text from the profile is used in the Print Dialog's 'Color Matching' panel. If the scripts can not be matched, then the string from the profile's name tag is used. See the 'Advanced Color Imaging on the Mac OS' book for more information on the 'dsc2' tag.

Intent Overrides

Added to LaserWriter 8.5.1's 'Color Matching' panel is a pop-up that allows the user to override the system profile's intent for a given print job. The intent pop-up lists the four possible intents (Perceptual, Relative Colorimetric, Saturation, and Absolute Colorimetric) as documented in the ICC Profile Specification.The pop-up also offers an automatic option which will cause the driver to use a Saturation intent when drawing text and line objects but to use the Perceptual intent on images.

PrGeneral

LaserWriter 8.5.1 adds support for the PrGeneral opcode (enableColorMatchingOp) for enabling and disabling color matching as described in 'Advanced Color Imaging on the Mac OS.' This PrGeneral opcode is only supported if the caller passes in an extended print record. If the caller passes in a standard print record, PrGeneral returns an error indicating that the opcode is not implemented. Enabling color matching via this call puts LaserWriter 8.5.1 into PostScript color matching mode with the printer's default Color Rendering Dictionary. This is the only device independent color matching mode available in the driver. If the caller disables color matching using PrGeneral, then the driver uses the Color/Gray scale color mode.

Destination Profile

LaserWriter 8.5.1 allows applications to determine the destination profile set by the user in the Color Matching panel. In order to get this information, an application must use the extended print record. After running the Print dialog, an application can use the PrGeneral kGetExtendedPrintRecOp (opcode 19) to obtain a Collection Manager collection describing the user's job parameters. Once this collection is obtained, the application uses the Collection Manager to obtain tag kHintDestProfileTag ('dprf'), id kHintDestProfileId (1). The data returned by the Collection Manager for this tag is the FSSpec of the user's chosen destination profile. If the tag does not exist in the collection, then the user has requested the printer's default Color Rendering Dictionary. For more information on collections, please see 'Inside Macintosh: QuickDraw GX Environment and Utilities.'

Faster First-Page Out

Today, when printing in two-pass mode with LaserWriter 8.4.x and below, all of a document's fonts are downloaded prior to the PostScript for the first page. In order to improve the printing time of the first page of a document, when printing in two-pass mode directly to a printer, LaserWriter 8.5.1 sacrifices page independence and downloads fonts immediately prior to the first

Collation

With LaserWriter 8.5.1, Apple has added collation to the driver. The general panel of the Print Dialog will add a collate check box if there is no collate option available in the PPD. With LaserWriter 8's collate option, when it is checked, and while printing n copies, the LaserWriter 8.5.1 driver will print the entire document n times in the following order: 1-2-3, 1-2-3, rather than 1-1, 2-2, 3-3. If the printer's PPD states that the printer supports collation itself, then the LaserWriter 8.5.1 driver is prevented from printing the job n times, and the collate option shows up in the 'Printer Specific' panel. In this case, the printer, not LaserWriter 8.5.1, is responsible for collating the output.

Media Selection

With LaserWriter 8.5.1, in order to support printers that use deferred media selection more fully, and to make it easier to print onto multiple media types, if a printer indicates that it supports media types (with the "*MediaType" keyword in the PPD), these types are listed in the Paper pop-ups in the Print Dialog.

Print to PDF

Due to many requests, in order to support the creation of Adobe Acrobat PDF documents, the Save panel of LaserWriter 8.5.1 adds Acrobat PDF as one of the supported formats. When Acrobat PDF is selected the PDF options are displayed in the save panel.

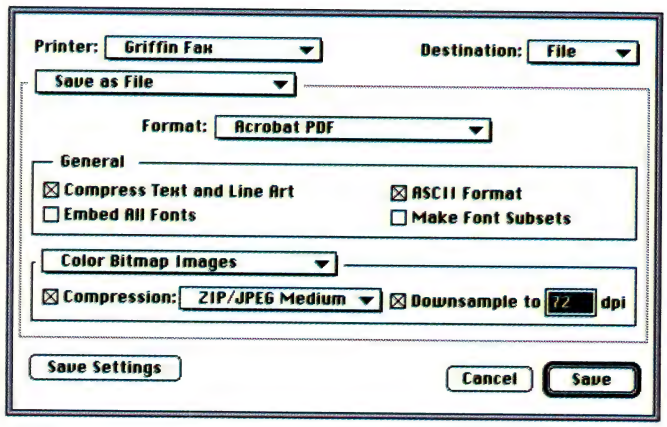


Figure 3.

When the user selects Acrobat PDF as the destination file format, LaserWriter 8.5.1 creates a PostScript file in the user's Temporary Items folder, and then launches Acrobat Distiller passing the location of the PostScript file and the intended location of the PDF file. The driver waits for Distiller to finish making the PDF file and then deletes the temporary PostScript file. This process is referred to as 'local distilling' as it requires Distiller to be run on the user's machine rather than on a remote server.

Because LaserWriter 8.5.1 uses Apple Events to communicate with Distiller, there is a requirement that the application running the driver's converter must be high-level event aware. In practice, this means that if an application is not high-level event aware, then foreground printing will be disabled if save to PDF is requested. In order to create PDF files from non-high level event aware applications, the user must print in the background.

Note: If Distiller is not available on the user's mounted hard drive, then the Acrobat Distiller option is disabled.

Summary

This article has described all of the new features of LaserWriter 8.5.1. We encourage users and developers to explore the new printing capabilities with their printer.

Further References

- Apple's TechNote web site <http://devworld.apple.com/dev/articles.shtml>.
- TN #5003: PPD 4.3 Specification from Adobe Systems Incorporated.
- PostScript Level 3 documentation from Adobe Systems Incorporated.
- Inside Macintosh: Imaging With QuickDraw.
- Inside Macintosh: Advanced Color Imaging on the Mac OS.
- Inside Macintosh: QuickDraw GX Environment and Utilities.
- ICC Profile Specification from <http://www.colororg.com>.

Acknowledgements

Thanks to Leticia Alarcón, John Blanchard, Richard Blanchard, Paul Danbold and David Gelphman.

Want to know what products are available for MacOS development? Check out Developer Depot™ <http://www.devdepot.com>

by Brendan Galten and Ingrid Kelly, Apple Computer, Inc.

Customizing Desktop Printer Utility

In the current LaserWriter 8 driver, the Chooser lists printers which are available on the AppleTalk network (either LocalTalk or AppleTalk) with an NBP type of "LaserWriter." When the user selects one of these printers, LaserWriter 8 creates a desktop printer by sending an Apple Event to the Finder. In the past, this mechanism of setting up printers has served the needs of most users. However, with LaserWriter 8.5.1, Apple is responding to several requests for more flexible desktop printing functionality. To this end, Apple has introduced a new application called Desktop Printer Utility which enables users to create additional types of desktop printers. Users can now create desktop printers which use the Unix lpr protocol for printing, in addition to the regular AppleTalk PAP printers. There are also "hold" printers which represent local print queues and "virtual" printers which represent printers which are not available on the network.

Along with the expanded capabilities, Desktop Printer Utility is also customizable. LaserWriter 8.5.1 and Desktop Printer Utility (together) support the creation and use of desktop printers known as Custom DTPs. When printed to, Custom DTPs cause the LaserWriter 8.5.1 driver to create a PostScript file and to launch an application that can post-process the PostScript. This post-processing application can do anything it likes with the PostScript file, such as converting the PostScript into another file format, transferring the file to another location using a modem or a network connection, or displaying the PostScript file to the user. This article describes how a developer might customize Desktop Printer Utility for use with an application.

Customizing the DTP Utility

In order to add a new custom DTP to the list of supported DTPs in Desktop Printer Utility, you must add resources to the utility's resource fork. Desktop Printer Utility contains templates to ease the creation of these resources.

'CsDs'(CustomDTPResource) Resource

In order to customize Desktop Printer Utility, a 'CsDs' (CustomDTPResource) resource must be added to Desktop Printer Utility. The LaserWriter uses this information to find the appropriate application to launch and to customize your DTP interface.

The resource's format is defined by the structure 'CustomAppDesc:'

```
#define kVariableLen 1

struct CustomAppDesc{
   OSType appSignature;
   Str255 docType;
   Str255 helpText;
   Str255 usage;
   Str255 appFileName;
   short numOfHintsFollow;
   HintRsrcSpec hintRsrc[kVariableLen];
};
typedef struct CustomAppDesc CustomAppDesc;
```

- appSignature: The application signature of the application to launch. The LaserWriter driver uses this information to find the application on all mounted volumes which are available to the user.
- docType: A string that describes the type of custom DTP. This string appears in the list of document types when "New" menu is selected. This string is highlighted in **Figure 1**.
- helpText: This string appears when your custom DTP is selected. This string will appear in the "New" dialog field just

Brendan Galten is a software engineer with RBI Software Systems in Berkeley, Ca. He resides in Petaluma, Ca. When not writing software, he enjoys spending time with his beautiful wife, Rosemary, and making sure his 9 month old daughter, Kelly, doesn't eat bugs. Camping, fishing and woodworking occupy what little time is left.

Ingrid Kelly is a technical support engineer at Apple Computer. She and her husband, Lance, are constantly in search of the best new restaurants in the San Francisco Bay Area. One day, they may decide to leave the computer industry altogether and open one of these great restaurants themselves. For now, though, they are content helping Silicon Valley grow.

below the DTP choices. This string appears as 'Custom DTP Example Help Text' in **Figure 1**.

- usage: A string that describes the usage of your custom DTP. This string appears as 'Custom DTP Example Usage string' in **Figure 2**.
- appFileName: The name of the application to launch. The LaserWriter driver uses this information to find the application on disks which are available to the user as well as to provide error messages to the user (e.g. "SurfWriter could not be launched due to insufficient memory").
- numOfHintsFollow: The number of hints provided in the hintRsrc array. See the 'Hints Resources' section for more details.
- hintRsrc: An array of hints that outlines the kind of PostScript that should be generated by the driver. See the 'Hints Resources' section for more details.

Figure 1.

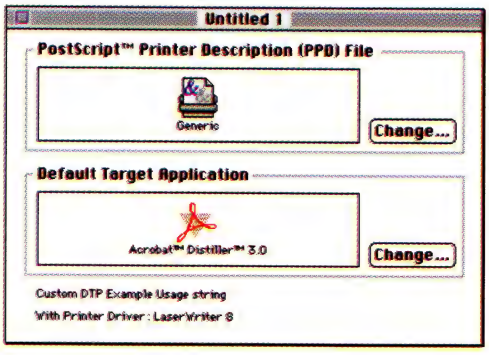


Figure 2.

Hints Resources (optional)

The CustomDTPResource can provide 'hints' about the PostScript the driver should generate. Each 'CustomDTPResource' contains the field 'numOfHintsFollow' which represents the number of 'hints' provided. The 'hints' data is placed in an array that contains 'numOfHintsFollow' number of copies of the structure:

```
struct HintRsrcSpec{
   OSType rsrcType;
   short rsrcID
};
typedef struct HintRsrcSpec HintRsrcSpec;
```

Each entry in the array points to a resource which is converted to a hint in the driver's preferences. The hint resource types, which are currently supported, have the following formats:

/* PostScript Level */

The application that postprocesses the PostScript for a custom DTP may require or prefer a certain PostScript level. The 'pslv' hint specifies the desired PostScript level.

```
Resource type 'PSlv'
typedef struct{
   OSType hintType;          //'pslv' hint type
   long hintId;              // set to 1
   long psLevel;
}PSlvResource;

#define kHintLanguageLevelTag 'pslv'
#define kHintLanguageLevelId  1
```

The possible(current) values for psLevel which can be set are:

```
-3,    Level 2 and 3
-2,    Level 1 and 2
-1,    Unknown Level
 0,    Other Level       //do not use
 1,    Level 1
 2,    Level 2
 3     Level 3
```

Note: -3 will create PostScript that takes advantage of PostScript level 2 and level 3 printers, but the data may fail on a level 1 printer. -2 will create PostScript that will succeed on level 1, 2, and 3 printers.

/* Binary OK - 'bnok' */

This hint allows or disallows the use of binary data in the PostScript file.

```
Resource type 'BNok'
typedef struct{
   OSType hintType;          //'bnok' hint type
   long hintId;              // set to 1
   TriState binaryOK;
}BNokResource;

#define kHintADOIsBinaryOKTag 'bnok'
#define kHintADOIsBinaryOKId  1
```

There are three acceptable settings for binaryOK which answer the question, "Is binary data ok?":

```
0,       False
1,       True
2        Unknown       //do not use
```

Note: This hint should not be created in the DTP with a value of unknown; however, it is available for future use.

/* Job Type - 'jobt' */

This hint specifies the type of PostScript to generate.

```
Resource type 'JObt'
typedef struct{
   OSType hintType;          //'jobt' hint type
   long hintId;              // set to 1
   char jobType;
}JObtResource;
```

```
#define kHintJobTypeTag     'jobt'
#define kHintJobTypeId      1
```

Currently only two PostScript forms for jobType are available:

```
0,  psJobPostScript
1   psJobEPSNoPreview
```

/* Font Handling - 'font' */

This hint specifies the font inclusion. The font handling hint consists of a four-byte flag followed by a list of NULL terminated font names.

```
Resource type 'FOnt'
typedef struct{
   OSType hintType;          //'font' hint type
   long hintId;              //set to 1
   long tag;
   unsigned char name[1];
}FOntResource;

#define kHintIncludeFontsTag      'font'
#define kHintIncludeFontsId          1
```

The possible values for tag are:

```
0,  kIncludeNoFontsOtherThan
1   kIncludeAllFontsBut
```

If the flag is 'kIncludeNoFontsOtherThan', the utility will only include the font definitions of fonts that are listed after the flag. If there are no font names after the flag, no font resources are included in the output. If the flag is 'kIncludeAllFontsBut', font resources are included in the output unless the font names are listed. To include all fonts, use the 'kIncludeAllFontsBut' flag with no fonts listed afterwards.

Sample Custom DTP

Following is a sample of a custom DTP that excludes Courier and Helvetica from the PostScript file via a 'CsDs' and 'FOnt' resource. This sample is intended to give developers an idea of how easy it is to customize the DTP Utility.

The first step towards adding the Custom DTP is to create a hint resource of the correct type. The example shown is the 'FOnt' resource which provides data about font inclusion.

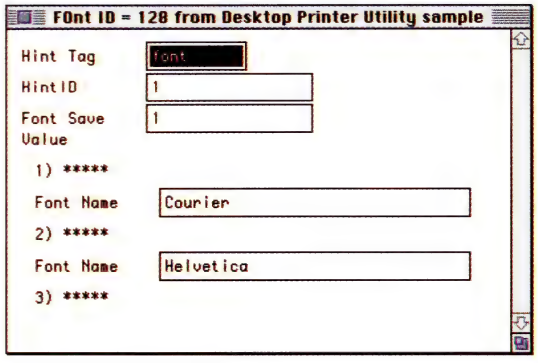


Figure 3.

The second step is to associate the hint with a particular DTP by adding hint resource information to the 'CsDs' resource for a particular DTP.

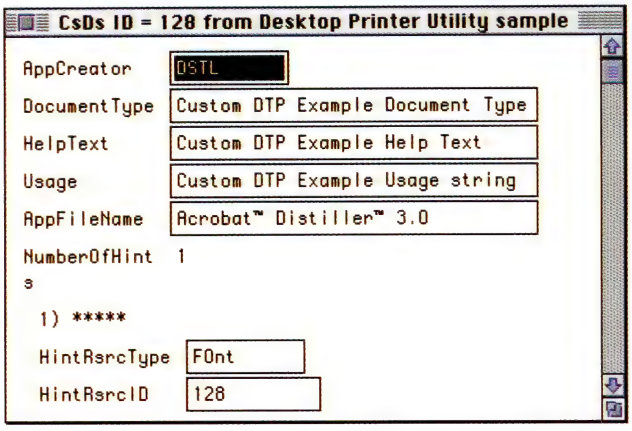


Figure 4.

And that's all that is required to get the DTP Utility to create a custom DTP that produces a PostScript file without Courier and Helvetica.

Limitations of the DTP Utility

Due to the parallel development schedules of Mac OS 8.0 and LaserWriter 8.5.1, DTP Utility 1.0 (which ships with LaserWriter 8.5.1) will not install on Mac OS 8.0. It will, however, work with Mac OS 8.1, due out in Fall of 1997.

Licensing the DTP Utility

In order to ship a customized DTP Utility with your application, you must license the DTP Utility from Apple. Please contact Apple's Software Licensing group (<sw.license@apple.com> or 512-919-2645) for more information.

Summary

This article has described and demonstrated all that is necessary for you to customize Desktop Printer Utility. We encourage all developers to give this a try.

Further References

- 'Introducing the LaserWriter Driver Version 8.5.1' by Ingrid Kelly in this issue of MacTech.

Acknowledgements

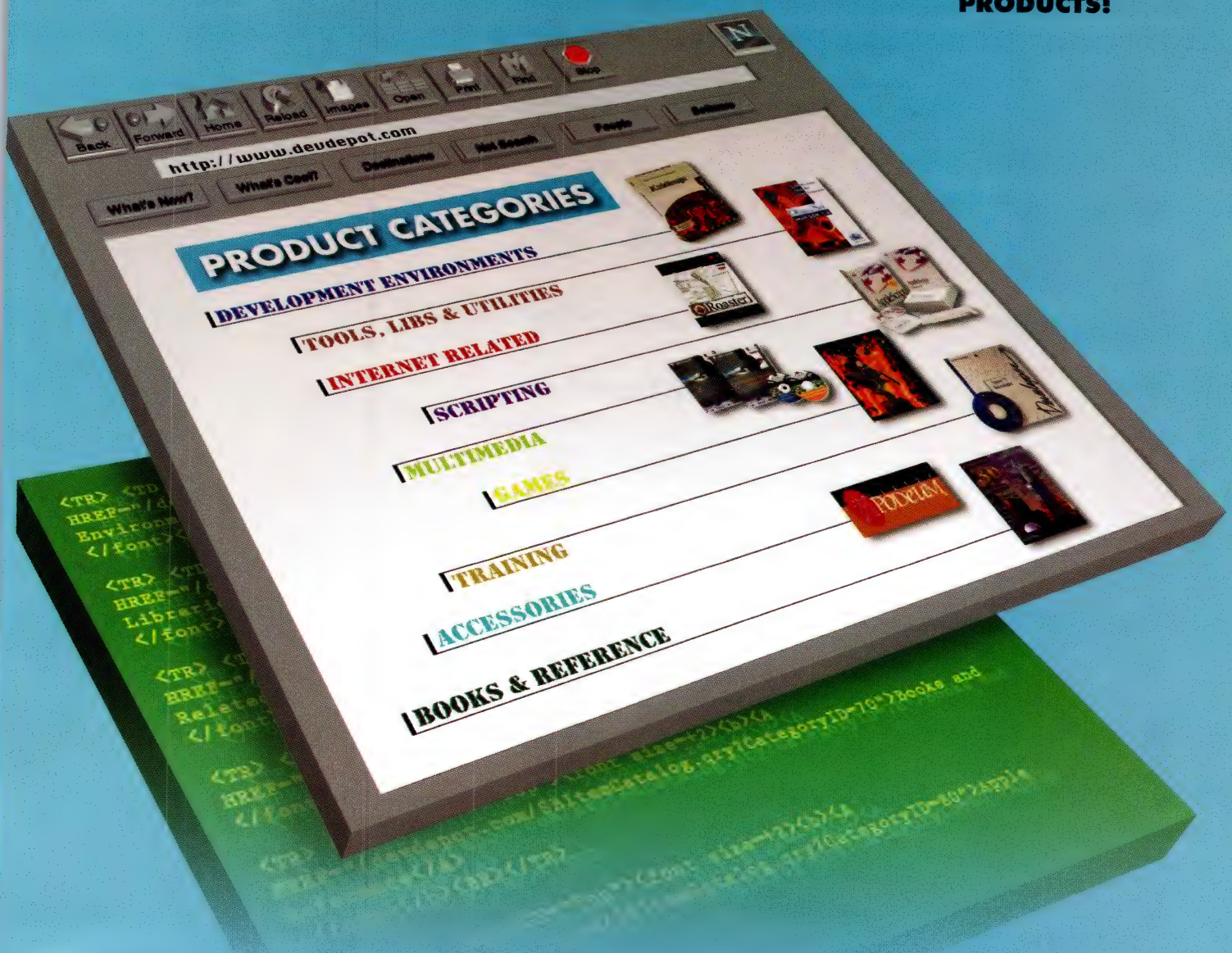
Thanks to Leticia Alarcon, John Blanchard, Richard Blanchard, Paul Danbold and David Gelphman. MT

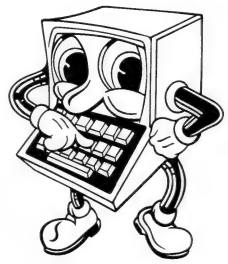
Developer DEPOT™

Developer Depot PO Box 5200 Westlake Village CA 91359-5200
800/MACDEV-1 • Outside the U.S. & Canada: 805/494-9797 • Fax: 805/494-9798
E-mail: orders@devdepot.com • Web Site: http://www.devdepot.com

- ✓ World renowned customer service
- ✓ Order by phone, E-mail, fax or through our continually updated Web site
- ✓ Hundreds of developer products
- ✓ Satisfaction and lowest price guaranteed

INCLUDES EVEN MORE APPLE BRANDED DEVELOPMENT PRODUCTS!





Developer To Do List:

- ☑ Prioritize project list & generate schedule
- ☐ Pick up new MacTech CD-ROM v. 1-12
- ☐ Check out all the Dev Depot specials
- ☐ See Apple Branded Tools at Dev Depot
- ☐ Tell the gang about Dev Depot deals
- ☐ Stock up on the latest books & utilities
- ☐ Enter Dev Depot info into contact file

Developer Depot
PO Box 5200
Westlake Village, CA 91359-5200

Voice: 800/MACDEV-1
Outside US/Canada: 805/494-9797
Fax: 805/494-9798
E-mail: orders@devdepot.com
http://www.devdepot.com

Developer DEPOT™

ANNOUNCEMENT

Developer DEPOT™

now carries Apple brand products

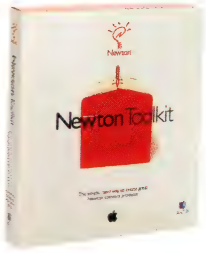


TABLE OF CONTENTS

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

One & 2-Day SHIPPING (call for details)

Order Toll-free 800-MACDEV-1 (800-622-3381)

Developer Depot 30 day Money Back, Price and Satisfaction Guarantee

Developer Depot products are sold with a 30 Day money back guarantee on user satisfaction, lower prices and against defects. If, for any reason, you are not satisfied or find the same product at a lower price within 30 days, please call Customer Service at 800-MACDEV-1 and request a Return Merchandise Authorization (RMA) number to get a full refund or the difference in price (where applicable). You must return undamaged product at your expense, including all its original packaging, documentation and the blank warranty card if applicable. Developer Depot will replace defective product upon receipt of the defective merchandise. Please remember to back up your data before installation of any new hardware, software, or peripherals; we cannot be responsible for any lost data. Policies, item availability, and prices are subject to change without notice. The price in effect when we receive your order will be the price that you are charged. We are not responsible for any typographical errors in this or any other catalog, nor for any misstatements from any vendor. Purchase orders are not accepted without prior approval. Call for more information.

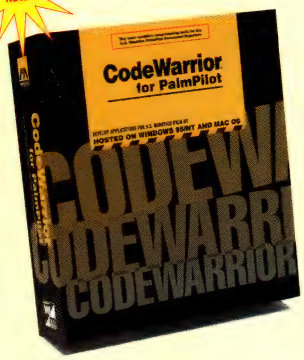
Stuff our lawyer made us write.
Developer Depot makes no other warranties. All other warranties, either expressed or implied, including the implied warranty of merchantability and fitness for a particular purpose are disclaimed. Developer Depot shall not be liable for any direct, special, incidental or consequential damages including lost profits, from any delay in delivery, or for any personal injury arising from the use of any product sold through Developer Depot. The limit of direct damages, if any, shall not exceed the purchase price of the product.© 1997 Xplain Corporation. All rights reserved. Any unauthorized duplication is in violation of federal laws. Developer Depot is a registered trademark of Xplain Corporation. All product names in this catalog are the trademarks of their respective holders.

© 1997 Xplain Corporation. All rights reserved. Any unauthorized duplication is in violation of federal laws. Developer Depot is a trademark of Xplain Corporation. All product names in this catalogue are trademarks or registered trademarks of their respective holders

Check out our Web site!
• Full product descriptions • Hundreds of more products
http://www.devdepot.com

metrowerks

NEW PRODUCT!



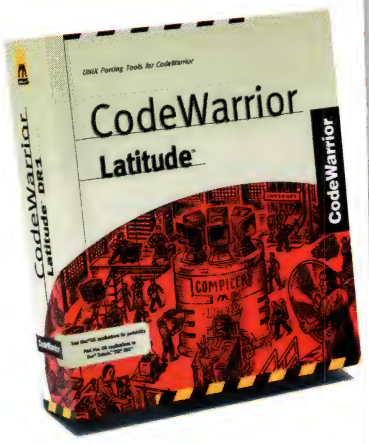
CodeWarrior for PalmPilot

by Metrowerks

CodeWarrior for PalmPilot is the first complete set of tools which enables you to develop for the U.S. Robotics PalmPilot connected organizer, from the comfort of your PC or Macintosh computer. All the tools you need are included: the award-winning CodeWarrior IDE, compiler, linker, source-level debugger, GUI builder, and other tools, plus online documentation and reference materials. Includes one free product update and free technical support with registration.

(SCWPALM) Our Price **$369**

NEW PRODUCT!



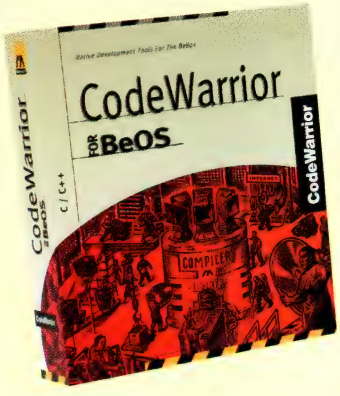
CodeWarrior Latitude

by Metrowerks

Don't throw away the investment you have made in your Mac OS application! With the new DR2 release of CodeWarrior Latitude, you can now port your Mac OS application to Rhapsody, as well as Silicon Graphics IRIX and Sun Solaris. At the heart of Latitude is a set of shared libraries which performs the functions of the Macintosh API. You recompile your Mac OS source code, linking it with the Latitude libraries to produce a native application. As the Rhapsody API evolves, so will Latitude. Registered users of CodeWarrior Latitude will receive all developer releases, the first full release, plus one additional product update, to ensure that you have access to the most up-to-date Rhapsody porting tools available.

(SCWLAT) Our Price **$399**

Be OS™



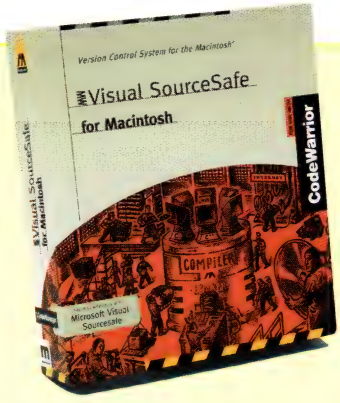
CodeWarrior for BeOS 3

by Metrowerks

- Start programming for the new, innovative Be Operating System (BeOS) with complete set of Codewarrior tools
- BeOS-native Integrated Development Environment (IDE) with all the familiar CodeWarrior features at your fingertips
- A BeOS PowerPC compiler and linker, an editor w/syntax color and styling, and a source-level debugger
- BeOS header and libraries, complete documentation, useful C++ classes, and sample code
- Now includes Java support
- The BeOS Preview Release allows you to run and program for the BeOS on a 603 or 604 PCI based PowerMac

(SCWFB) Our Price **$299**

NEW PRODUCT!



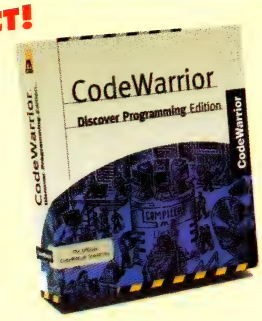
MW Visual SourceSafe Release 5

by Metrowerks

- Source code control system, plug-in to the CodeWarrior IDE or stand-alone Client application
- Compatible with Microsoft Visual SourceSafe version 5.0
- Cross-platform support (Mac, Windows, UNIX)
- Configuration management in excess of 4 billion files
- Over 8,000 files and sub-projects in a single sub-project
- Registered users receive one year of free technical support and one free update

(SMWVSS) Our Price **$499**

NEW PRODUCT!



CodeWarrior Discover Programming Edition

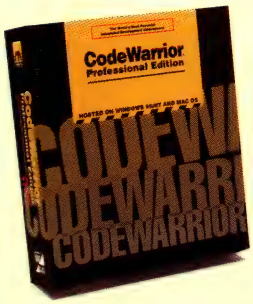
by Metrowerks

- Learn to program in C/C++/Java/Pascal
- Full-featured programming product - not a "lite" edition
- Online books and tutorials
- Hosted on either Mac OS or Windows 95/NT

CodeWarrior Discover Programming Edition offers an unbeatable combination of full-featured programming tools, online books and instructional materials, all at a great price. Whether you want to learn the basics of programming or add a new language to your skill set, Discover Programming puts the information and tools you need at your fingertips: the award-winning, easy-to-use CodeWarrior Integrated Development Environment (IDE), four of the most popular programming languages, online books, online tutorials, sample source code and free technical support (with your registration). Discover what you can learn with CodeWarrior, buy your copy today!

(SDPED) Our Price **$79**

Order Toll-free
800-MACDEV-1
(800-622-3381)



NEW PRODUCT!

CodeWarrior Professional Release 2

by Metrowerks

- Includes Windows 95/NT and Mac OS versions of the CodeWarrior IDE
- Supports C, C++, Java and Pascal
- Develop for Windows 95/NT on x86 and Mac OS on 68K/PowerPC
- V2 Project Manager supports multiple open projects, subprojects, multiple targets per project, and threaded execution
- Support for JDK 1.1.3 in CodeWarrior Java

More platforms, more languages, more options: CodeWarrior Professional is designed to give you the tools you need for serious, industrial-strength programming. CodeWarrior Professional is the only Integrated Development Environment (IDE) in which you can edit, compile and debug C, C++, Java and Pascal programs for multiple target processors and operating systems. CodeWarrior's compilers produce fast, highly optimized code for Windows 95/NT running on x86, or Mac OS running on 68K or PowerPC processors. CodeWarrior Professional features the CodeWarrior IDE Version 2. The revamped Project Manager supports multiple projects open simultaneously, multiple targets per project, and threaded execution, and it's significantly faster. CodeWarrior Professional also includes online books, documentation, and reference materials, as well as tutorials and sample code. We support your development efforts with one free update and free world-class technical support for a year with registration.

Upgrade for competitive product owners

(SCWPRUP) Our Price **$449**

Full Version

(SCWPRO) Our Price **$599**

Apple Dylan
dynamic object oriented language and development environment

Apple Dylan Technology Release

by Apple Computer, Inc.

- Contains a PowerPC-native prototype version of a development environment based on the Object Oriented Dynamic Language (OODL) Dylan. Developers will be able to produce code targeting both 680x0 and Power Macintosh systems
- Automatic memory management
- Application framework and user-interface & builder
- High-level exception handling
- Cross-language support for C code and APIs

CD & Online Documentation

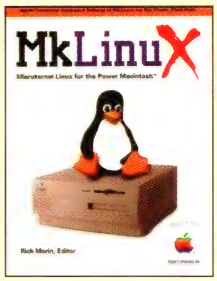
(SADTRO) Our Price **$39.95**

CD & Hardcopy

(SADTRH) Our Price **$59.95**

Lowest price guaranteed.
Feel Good about your purchase!
30-day return policy.

Check out our Web site!
• Full product descriptions • Hundreds of more products
http://www.devdepot.com



NEW PRODUCT!

MkLinux: Microkernel Linux for the Power Macintosh

by Prime Time Freeware

MkLinux is a native port of Mach 3 and the Linux 2.0 kernel, complemented by hundreds of commands from BSD, GNU, and X11. It runs on most (NuBus abd OCU bus) Power Macintosh systems; Performa, PowerBook, and multiprocessor ports are currently under development.
MkLinux is robust, powerful, freely distributable, and source code compatible with most other Linux systems. It provides a full suite of development tools, support for AppleTalk, HFS, and Objective-C, and access to a vast amount of free software. MkLinux is a great way to "come up to speed" on Mach, UNIX, and Rhapsody.

- MkLinux user community supports FTP and web servers, development and porting efforts, and several mailing lists
- The Apple sponsored reference release contains a wealth of introductory and reference material on Linux, Mach, NeXT, and the Power Macintosh
- Includes free 3.0 upgrade

(BMKLINUX) Our Price **$50**

Pro Fortran

by Absoft Corporation

Absoft Pro Fortran combines native F90, VAX compatible F77, and C/C++ compilers into a single, easy to use environment. All compilers are link compatible and operate through a common interface.

- Graphical debugger, browsers, array display, performance profiler, linker, MRWE application mainframe
- MIG graphics library, Absoft Create Make, several utilities, the latest version of MPW and illustrated documentation
- Whole array operations, modules, interface blocks, and user-defined types or data structures
- Dynamic memory allocation and new control constructs
- F90 is link compatible with Absoft F77, C++, MrC and CodeWarrior
- It is fully compatible with Toolbox, MPW tools, and most third-party products

(SAPROF) Our Price **$899**

Order Toll-free
800-MACDEV-1
(800-622-3381)

VIP–BASIC: Visual Interactive Programming in BASIC

by Mainstay

Now you can create full-featured, stand–alone Macintosh and Power Macintosh applications in standard BASIC code! VIP-BASIC 2.0 is the fastest way to program your Macintosh.

- Rapid application development environment with application framework, mix and match: VIP–BASIC high-level subprograms
- Import pre-existing BASIC code: automatically integrate BASIC code, export C Code for compiling: automatically convert your BASIC code to C for compilation with Metrowerks' CodeWarrior

(SVIPBASIC) Our Price **$195**

VIP–C: Visual Interactive Programming in C

by Mainstay

Now you can create full-featured, stand-alone Macintosh and Power Macintosh applications in just minutes. VIP-C 2.0 is the first rapid application development system for creating complete Macintosh programs in standard ANSI C.

- Includes powerful, tightly integrated visual debugger, Import pre-existing C code: automatically integrate C code with a current project
- Includes full-featured mini database: (ltd to 32K) of the powerful VIP–BASIC database manager gives you everything you need to setup royalty–free, multi–user database applications

(SVIPC) Our Price **$295**

Macintosh Common Lisp 4.0

by Digitool, Inc.

Macintosh Common Lisp provides users with a rich set of object-oriented dynamic language features making it especially well-suited for rapid prototyping, custom development for business and education, scientific and engineering applications, and academic research.

- Power PC native environment & compiler, full Macintosh support
- CLOS, the standard Common Lisp object system
- Interactive dynamic environment, multiple processes
- Automatic memory management and self-typing data
- Ephemeral garbage collector, smaller application footprint
- Compiles with Common Lisp industry standard and smart programmable tools, 110+ mb of user contributed code
- Complete on-line documentation (manual sold separately)
- Software license and registration card

(SMCLISP) Our Price **$675**

ObjectMaster Professional Edition

NEW PRODUCT!

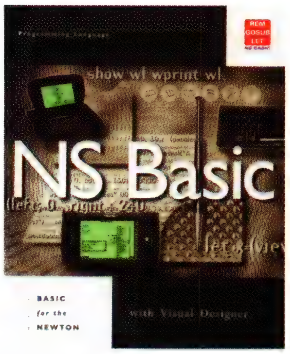
by Altura Software, Inc.

Object Master is an innovative programming environment that provides all the necessary tools to write, organize, and navigate through source code.

- Write code using the most robust source code editor available on the desktop
- Organize source code into projects to quickly access and manipulate all files
- Navigate through source code using intuitive graphical Browser windows

(SOMPE) Our Price **$399**

NS BASIC 3.6 for the Newton with Visual Designer

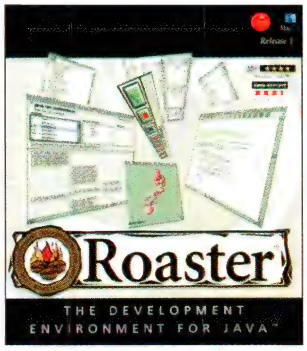


by NS BASIC Corporation

- A fully interactive implementation of BASIC programming language
- Runs entirely on the Newton – no host is required
- Create files, access the built in soups, and the serial port for input and output
- Work directly on the Newton, or through a connected Mac/PC and keyboard
- Get the **BASIC Internet Tool**, available at no charge to NS BASIC users from www.nsbasic.com
- Release Notes with sample code are available from the same location
- Runs on any Newton MessagePad 130 with NS BASIC and the Newton Internet Enabler. Also runs on MP 1201s with NOS 2.0 that have full memory available
- Write short programs to access News, mail and the web

(SNSBASIC) Our Price **$99**

Roaster



by Roaster Technologies, Inc.

Get the most out of Sun's Java™ programming language with this powerful development environment.

- Features include: ability to build stand-alone Macintosh applications or applets; visual interface builder; ability to create cross-platform zip files; powerful Java debugger; wizard for quickly creating Java applets or applications; JDBC included; Java object database included; ability to call AppleScripts from Java; Just-in-time (JIT) compiler; JDK 1.0.2 support; class tree and hierarchical class browser; much more!
- Software includes: Over 300 example applets and applications; Netscape Internet Foundation Classes; Object Design's PSE for Java; OpenLink Software's JDBC Drivers; OpenSpace Java Generic Library; Microline Component Toolkit Lite 3.0; much more!
- Requirements: Runs on 68k or PowerPC, CD-ROM
- Price includes all web-based updates

(SROAST3) Our Price **$99**

TENON INTERSYSTEMS

CodeBuilder

by Tenon Intersystems

CodeBuilder is a powerful and unique Macintosh software development tool for porting existing apps or developing new, advanced applications on Power Macs and Power Mac clones.

- A powerful Macintosh software development tool suite of C, C++, Objective-C, Java, Ada, and Fortran development tools
- Complete UNIX & X development environment for developing UNIX or Macintosh apps
- Includes compilers and source-code debugger for Objective C, and C, C++, Ada 95 and Fortran 77
- Web & internet scripting tools: Perl, MacPerl, tcl/tk, bash, sh, and csh
- Supports Rhapsody kernel APIs and Rhapsody TCP sockets

(SMIOCODEB) Our Price **$149**

WAIT... There's More!

Here's a list of all available products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

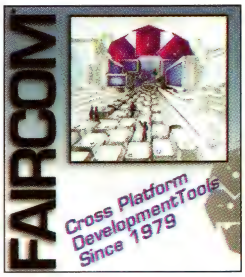
PRODUCT	CODE	OUR PRICE
LPA MacProlog Developers Edition	SLPAD	995.00
LPA MacProlog Programmers Edition	SLPAP	495.00
LS Fortran Pro	SLSFORT	595.00
LS Fortran Plug-In	SLSFPI	199.00
Mac FORTRAN II	SFORT2	595.00
Power MachTen–UNIX	SM10PPC	695.00
Presenting Magic Cap	BPRESMAGIC	15.25
SmalltalkAgents	SSTA	695.00
Think Pascal 4.0	SPASCAL	165.00

ObjectSet Mail SDK
by Smartcode Software

- Powerful C++ classes for integrating Internet e-mail in your applications
- Helps you write software that can share mail with other leading e-mail products
- Royalty-free MIME, SMTP, and POP3 APIs
- Gives you the most robust MIME parser and encoder available
- Ideal for use in Internet and Intranet environments
- Comes complete with samples with documented, reusable source code
- Free standard technical support

(SOSMSDK) Our Price **$495**

r-tree Report Generator
by Faircom



Handles virtually every aspect of report generation:

- Complete C source
- Complex multi-line reports
- Multi-file access
- Complete layout control
- Conditional page breaks
- Nested Headers and Footers
- Horizontal Repeats

(SRTRG) Our Price **$445**

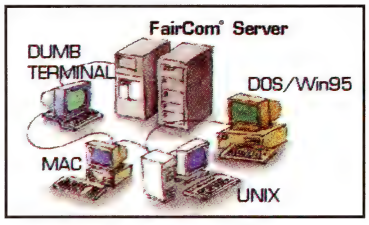
Spellswell Plus 2.1
by Working Software

- Award-winning, comprehensive, practical spelling checker that works in batch mode or within applications that incorporate the Apple Events Word Services protocol (e.g., Eudora, WordPerfect, Communicate!, and InfoDepot)
- Checks for spelling errors as well as common typos like capitalization errors, spaces before punctuation, double word errors, abbreviation errors, a/an before vowel/consonant, etc.
- MacTech orders include developer kit with Writeswell Jr., a sample AppleEvents Word Services word-processor and its source code
- Available for OEM Sales

(SSPELL) Our Price **$49**

c-tree Plus® Database Handler
by Faircom

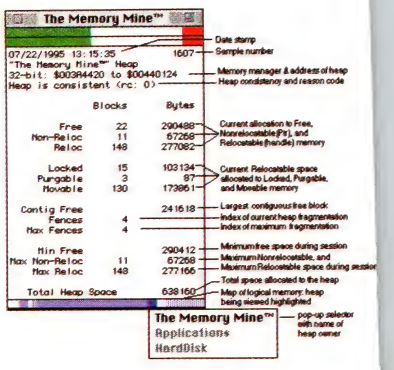
Unsurpassed Cross Platform Tools for Mac Developers!



- Full C Source
- Client/Server Option
- Over 16 years proven reliability
- Concurrent simultaneous access of Mac/PC files
- Superior throughput and performance
- Unparalleled scalability and flexibility
- Fixed/Variable length files

(SCTPDH) Our Price **$895**

Memory Mine
by Adianta Inc.



- Monitor heaps, identify problems such as memory leaks, and stress test applications
- Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is shown, as are heap corruption, fragmentation, and more
- Allocate, Purge, Compact, and Zap memory lets users stress test all or part of a program

(SMEMMINE) Our Price **$99**

NEW PRODUCT!

ScriptDemon
by Royal Software, Inc.

ScriptDemon is a browser plug-in that, for the first time, allows you to deliver and run AppleScripts from Web pages. The ScriptDemon plug-in will execute the embedded AppleScript code included on a Web page. ScriptDemon painlessly and inexpensively handles many previously impossible tasks, such as:

- Using the Intranet to manage all Macintosh computers on-line
- Using the Internet to install and configure software
- Using the Internet to configure hardware
- Delivering complex sets of files and assembling them on the browsing computer
- Providing interactive education and product support on both the Internet and Intranet
- A perfect companion to LiveCard!

(SSDEMON) Our Price **$949**

Guide Composer™ 1.2
by StepUp Software

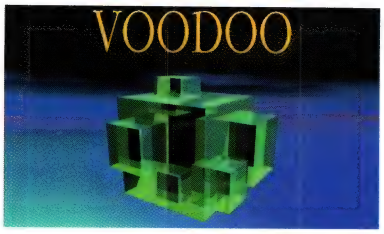
STEPUP SOFTWARE

- Create powerful Apple Guide help systems for any new or existing Macintosh application
- Provides a WYSIWYG development environment: Guide content is developed in Guide windows
- Design topics, phrases, and panels in the same format as the user will use them
- Features are WYSIWYG interface, Topics, phrases, and hierarchical phrases, Coach marks, Fully-Integrated with Apple's Guide Maker (distributed with Guide Composer), compiles scripts automatically, PICTs in Panels, Generated Guide scripts are modifiable
- FREE Update to all registered Guide Composer users. Demo is available at http://www.guideworks.com/

(SGCOMP) Our Price **$99**

SEE RELATED PRODUCTS: AppleGuide Complete, Danny Goodman's AppleGuide Starter Kit, Real World AppleGuide

VOODOO 1.8
by UNI SOFTWARE PLUS



- Stand-alone version control tool for all sorts of projects (software development, documentation, design, CAD, publishing, etc.)
- Smooth integration with Metrowerks CodeWarrior IDE
- Simple and clear management of variants and revisions of entire projects (not only of single files)
- Easy-to-use graphical project browser gives access to all versions that were ever stored.
- Recording of the complete history (who made which changes when and why)
- View differences between versions (not only for text files!)
- Efficient delta storage of arbitrary files (text as well as non-text files) gains savings of 95 % and more
- Administration of users with hierarchical access rights
- Configurable local file locking (Finder flag or 'ckid' resource)
- Scriptable, essential parts PowerPC native

Single license (SVOODO01) **$229**
2 pack (SVOODO02) **$359**
5 pack (SVOODO05) **$799**
10 pack (SVOODO10) **$1369**
20 pack (SVOODO20) **$2399**

Additional pricing available on request.
SEE RELATED CATEGORY: Dev. Environments

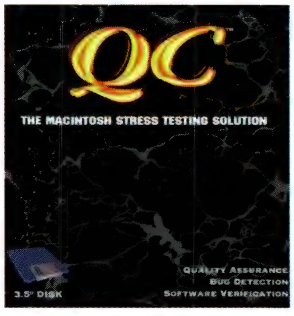
StoneTable 68K/PPC
by StoneTablet Publishing

StoneTable is a powerful and professional replacement for the List Manager used by developers worldwide. Version 3.0 is a new release with many improvements including better clipboard and drag/drop integration with other applications.

- Available for use with CodeWarrior C & Pascal
- Includes libraries for 68K (A4 & A5) and PowerPC
- An LTable-like class is provided to incorporate StoneTable into the PowerPlant environment

(SSTONEFAT) Our Price **$199**

QC
by Onyx Technology, Inc.



High performance runtime stress testing for applications.

- Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking
- Extremely user friendly – ideal for non-programmer testers
- Also available in Japanese

(SQC) Our Price **$99**

Spotlight
by Onyx Technology, Inc.

NEW PRODUCT!

Spotlight is a stand alone debugging aid that performs memory protection (arrays, heap accesses, outside your heap, low mem, etc), discipline checking on toolbox calls, and leaks detection.

- Spotlight is sold on an annual subscription basis
- The subscription service provides all updates
- Includes maintenance releases for one year after the initial purchase or renewal date.

(SSPTLT) Our Price **$199**

AppSketcher 1.0 for BeOS
by BeatWare, Inc.

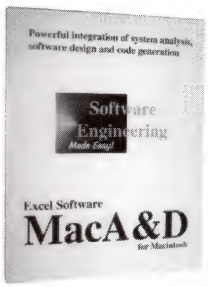
BeatWare

AppSketcher is the premier programming tool for the BeOS.

- The fastest way to develop software on the BeOS
- Drag and drop design is great for creating user interfaces
- Supports localization for worldwide sales
- Expands applications easily without manual code modifications
- Shortens development time by automating the Make process through direct communication with the BeIDE

(SASFB) Our Price **$199**

MacA&D 6.0
by Excel Software



Order Toll-free
800-MACDEV-1
(800-622-3381)

- Structured analysis and design
- Object-oriented analysis and design
- Real-time and multi-task design
- Data and screen modeling
- Integrated code editing and browsing
- Multi-user dictionary and requirements
- Code to design diagrams for C, C++,etc.
- Design diagrams to code for C, C++, etc.
- State modeling diagrams and tables
- Use cases with traceability

(SMACADP) Our Price **$1995**

Apprentice 6
by Celestin Company

Celestin

Apprentice 6 is a high-quality CD-ROM collection of over 600 megabytes of up-to-date source code, utilities, and info for Mac programmers. All of the source code and utilities are completely new or updated for this release.

- Frontier 4.1, the highly-acclaimed scripting environment
- More PowerPlant AND many more PowerPC samples
- Cool new languages and environments added (Clean, Eiffel, F, Tcl-Tk)
- Hot new demos from leading Mac development companies

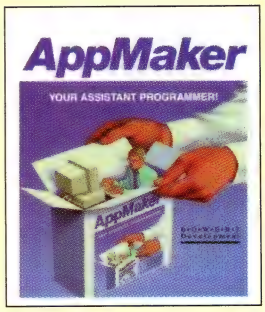
(SAPPRENT6) Our Price **$35**

SoftPolish CD-ROM
by Bare Bones Software

- The essential tool for software quality assurance on the Macintosh
- Helps you identify inconsistencies with Apple's user interface guidelines, misspelled words, missing resources, and other mistakes
- Provides tools to put the finishing touches on software distribution packages prior to release
- Works independently of any programming language or environment
- Ideal for sanity checking software throughout the development process

(SSOFTPOL) Our price **$99**

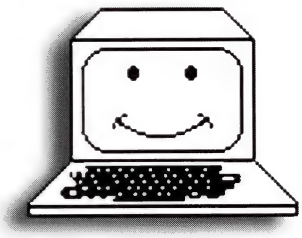
AppMaker
by Bowers Development



- Develop the user interface for a Macintosh application using the original interface builder
- Just point and click to design your application
- Creates resources and generates excellent source code
- Supports most development environments including Metrowerks, Symantec, or MPW; C, C++, or Pascal; procedural or object-oriented, using PowerPlant, TCL, or MacApp
- The generated code uses the Universal Headers to provide PowerMac compatibility
- Great tool for beginners to learn object-oriented and Macintosh Toolbox programming techniques
- Includes one-year subscription on CD and hardcopy documentation

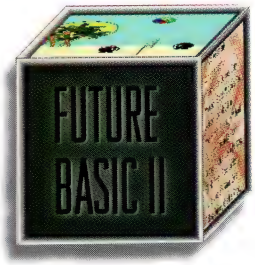
(SAPPMAKE) Our Price **$299**

B-Tree HELPER 2.2
by Magreeable Software



- Inexpensive database engine for Macintosh programmers in C source code
- Uses contiguous fixed length blocks
- Expands the file as necessary and contracts files when possible
- Inserts and deletes keys in one or more B-Trees
- Finds keys equal to, less than, or greater than a given value in a few hundredths of a second
- Finds lists of records whose keys are equal to, less than, or greater than a given value or are in a range of values

(SBTREE) Our Price **$149**



Future Basic II
by Staz Software

FutureBASIC II is the award winning leader in Macintosh BASIC programming.

- Source level debugger and Interactive compiler/editor
- Multi-file Project manager and Multi-file find and replace
- Super fast compilation, 32 bit clean, and System 7.x savvy
- QuickBASIC converter
- Getting Started manual with over 500 example files
- Full support of standard BASIC

(SFBASIC2) Our Price **$229**

dtF
by dtF Americas

- True relational database system for Apple Macintosh computers
- Provides a powerful choice for developers who want to create database centered applications with no performance trade-offs
- Features SQL, full transaction control, error recovery, single user, client server architecture and multi-platform support including DOS, Windows, OS/2 and UNIX
- The C/C++ API is identical and fully portable across all supported platforms
- Third-party vendors supporting dtF will be able to offer a variety of advanced features and benefits to their customers royalty free
- Tools are included for importing, exporting, creating and managing databases and users
- Supported development environments include: Symantec, MPW, Metrowerks and more Mac/SDK

(SDTF) Our Price **$695**

Step-Up Installer Pack
by StepUp Software

- Package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts
- Compression formats supported are Compact Pro & Diamond
- Each atom also available separately
- Compression requires additional licensing

(SINSTALL) Our Price **$219**

ScriptGen Pro
by StepUp Software

- Installer script generator which requires no programming or knowledge of Rez
- Supports StepUp's InstallerPack, StuffIt decompression, Compact Pro decompression, custom packages, splash screens, network installs, and resource installation

(SSCRPTGEN) Our Price **$169**

Water's Edge Software

Tools Plus libraries + framework

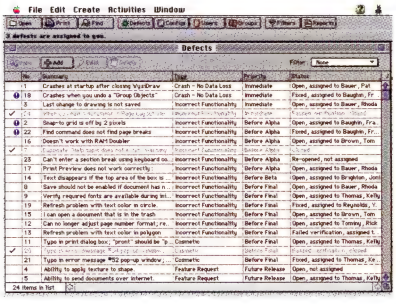
by Water's Edge Software

Easily create compact, fast running, professional looking applications and plug-ins*. Tools Plus lets you create virtually any user interface element with a single routine, and it transparently provides a robust infrastructure to make all your pieces work together as an application. Rated 4 stars by Macworld! MacTech (July 96): "it's an incredibly rich collection of tools... If you are interested in developing applications that have 'quality' written all over them, then Tools Plus is for you."

- Simplifies programming and thins source code
- Automates all standard GUI elements
- Thousands of extras, from floating palettes and tool bars to powerful picture buttons
- Includes numerous 3D grayscale options
- Over 1/2 MB of custom fonts, icons, cursors, and other resources
- Includes SuperCDEFs world-class controls (an $89 value) free

(STOOLCW) Our Price **$249**
CodeWarrior Gold (C/C++ & Pascal, 68K & PPC)
(STOOLCWB) Our Price **$199**
CodeWarrior Bronze (C/C++ & Pascal, 68K)
(STOOLSYMT) Our Price **$199**
Symantec (THINK) C/C++ and THINK Pascal (68K)
(STOOLSYM) Our Price **$149**
Symantec (THINK) C/C++ (68K)
(STOOLPAS) Our Price **$149**
THINK Pascal (68K)

*CodeWarrior required to write plug-ins



Order Toll-free
800-MACDEV-1
(800-622-3381)

TestTrack-Bug Tracking the Macintosh Way

by Seapine Software, Inc.

- Tracks bugs, feature requests, test configurations, users, and more
- Includes notifications, security, a powerful filter mechanism, and multiple reports
- Links your testers, engineers, documentations staff, and project managers together to ensure all bugs are identified, fixed, and documented
- Eliminates the need to build custom bug tracking solutions using general purpose database tools
- Supports single- and multi-user bug databases (additional licenses required to use multi-user features)

(STETR) Our Price **$129**

CodeBuilder

by Tenon Intersystems

TENON INTERSYSTEMS

CodeBuilder is a powerful and unique Macintosh software development tool for porting existing apps or developing new, advanced applications on Power Macs and Power Mac clones.

- A powerful Macintosh software development tool suite of C, C++, Objective-C, Java, Ada, and Fortran development tools.
- Complete UNIX & X development environment for developing UNIX or Macintosh apps
- Includes compilers and source-code debugger for Objective C, and C, C++, Ada 95 and Fortran 77
- Web & internet scripting tools: Perl, MacPerl, tcl/tk, bash, sh, and csh
- Supports Rhapsody kernel APIs and Rhapsody TCP sockets

(SMIOCODEB) Our Price **$149**

Phyla™: Object–Oriented Database

by Mainstay

- Powerful Databases Without Programming: Phyla handles your all your complex database needs
- Define a Database in Minutes: Using an intuitive, graphical user interface
- Objects Are a More Natural Approach: Phyla creates real world databases
- Drag–and–Drop Ease: Relate objects by simply dragging objects between windows
- Create Custom Forms and Reports: Quickly create custom forms and reports
- Fast Finds and Sorts: Perform complex queries and calculations without programming
- Synchronize Multiple Databases Copies
- Password Protection With Access Limitations
- Easy Import and Export: Import from other databases, export data in various formats

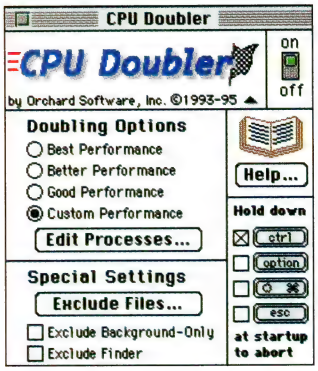
(SPHYLA) Our Price **$179**

BBEdit 4.5

by Bare Bones Software

BBEdit 4.5 is a powerful, easy-to-learn text and HTML editor that offers developers and HTML authors the ability to build on its core functionality to suite their specific needs through its plug-in architecture and scripting capabilites. This new version includes: a visual table tool that speeds page and site development, contextual menu support for Mac OS 8, improved storage for 'grep' patterns, scriptable HTML authoring preferences and more. It still provides: unparalleled searching muscle with support for both 'grep' style and advanced literal searches, the ability to quickly compare differences between files or entire folders, integrated support for Symantec's IDE, Metrowerks CodeWarrior, THINK Reference 2.x, MPW Toolserver and most other environments and a heck of a lot more.

(SBBEDIT) Our Price **$119**



CPU Doubler
by Orchard Software

- Performance enhancement utility for the Macintosh
- Increases the speed of your computer by 100%
- Works on both the PowerPC and 68K Macintosh
- Manages computer throughput using a proprietary scheduling algorithm
- Ensure optimal performance and compatibility

(SCPU2X) Our Price **$79**

SEE RELATED PRODUCTS:
Development Environments

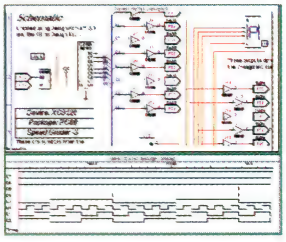
CompileIt!
by Royal Software, Inc.

NEW PRODUCT!

CompileIt!, the first HyperTalk compiler, is a complete developement system for the creation of XCMDs and XFCNs.

- Expand the capabilities of your environment by using CompileIt! and the ROM Toolbox extensions
- Increase the speed of routines written in HyperTalk by turning scripts into externals
- Protect sensative code from prying eyes because your code is now compiled!
- Easily learn Macintosh programming by exploring the ROM Toolbox
- Includes DebugIt!, a valuable source-level debugger for externals created with CompileIt!

(SCOMPIT) Our Price **$149**

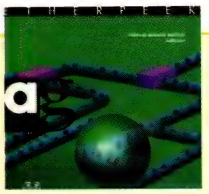


DesignWorks 4.0
by Capilano Computing

DesignWorks 4.0 has all the ease of use and schematic editing power of previous versions, plus new features designed to make your entire design process easier and more error free. The 4.0 version has new menu customization and scripting features that will directly address your design checking and interfacing needs.

- Flexible schematic editing features speed the drawing process
- Full Undo/Redo on all editing operations
- Hierarchical design with unlimited levels is fully supported
- Powerful attribute features allow arbitrary text information to be associated with any signal, device or device pin
- Extensive symbol libraries with over 12,000 parts in ANSI and IEEE format
- Integrated device symbol editor allows you to create custom symbols using standard drawing tools
- Interactive digital simulator option is available. No netlists, no application switching!

(SDWORKS) Our Price **$995**



EtherPeek
by AG Group, Inc.

NEW PRODUCT!

EtherPeek for Macintosh is a full-featured protocol analyzer that allows you to quickly and easily test and debug network communication, and:

- Check for protocol compliance
- Use hundreds of built-in decodes
- Develop custom packet decoders
- Filter packets during or after capture
- Test device reactions to specific packet types
- Customize or alter packets for transmission
- Generate traffic to test varying loads

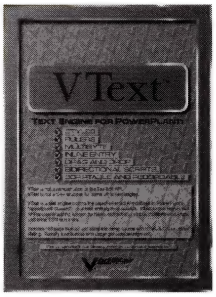
(SEPEEK) Our Price **$745**

OpenGL for the Macintosh
by Conix Graphics

OpenGL is the premier 3D graphics library that allows software developers the ability to develop high-quality, interactive 2D and 3D graphics applications. OpenGL can perform the following wide range of functions which will enhance the development of all graphics software:

- Geometric primitives (points, lines, and polygons)
- RGBA or color index mode
- Viewing and modeling transformations
- Texture Mapping, Lighting, Shading and Z Buffering
- Atmospheric Effects (fog, smoke, and haze)
- Alpha Blending (transparency)
- Antialiasing, Accumulation Buffer, Stencil Planes
- Display list or immediate mode
- Polynomial Evaluators (to support Non-uniform rational B-splines)
- Feedback, Selection, and Picking Raster primitives (bitmaps and pixel rectangles)
- Pixel Operations (storing, transforming, mapping, zooming)

(SOPENGL) Our Price **$389**

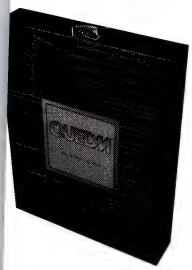


VText
by Vivistar

VText is a C++ add-on library for Metrowerks' PowerPlant application framework. VText provides complete Macintosh text support including: greater than 32kb text, undo, drag and drop editing, AppleEvent scripting and recordability, full support for multibyte characters and inline input methods including Japanese and Chinese text, and full support for bi-directional script systems including Arabic and Hebrew.

- Full featured text engine for Metrowerks' PowerPlant
- Stylesets and rulersets with tabs
- Flexible object oriented C++ API
- Full undo and drag and drop editing
- WorldScript savvy including bidirectional and multibyte scripts with inline editing
- AppleEvent factored for scriptability and recordability

(SVTEXT) Our Price **$349**



QUED/M 3.0

by Nisus Software

- The programmer's text editor that defined the industry standard for speed and efficiency
- PowerPC native
- Features integrated support for Symantec C/C++, Metrowerks CodeWarrior 6, and MPW
- Supports all the major development environments on the Macintosh.
- Powerful editing features, including unlimited undo and redo, macro language, scripting, text folding, ten editable/appendable clipboards, markers, displaying text as ASCII codes, dynamic coloring of C/C++ keywords/comments, rectangular and non-contiguous selection
- Includes Celestin Company's APPRENTICE 4

(SQUEDM) Our Price **$89**

AG Author

by Lakewood Software

AG Author 1.0 is a full-featured Apple Guide authoring tool with fully customizable project template. The following features are unique to AG Author:

- Support for styled, colored, & hot text
- Fully customizable project template
- Flexible compile options
- Find & replace tool for scripts
- Multiple open projects
- Rapid deployment of project globals

(SAGA) Our Price **$99**

NEW PRODUCT!

SEE RELATED PRODUCTS: AppleGuide Complete, Danny Goodman's AppleGuide Starter Kit, Real World AppleGuide

Bee-one

by Power Box

Bee-one lightens your load on the road by adapting relational databases developed under 4D© to the Newton Platform. Once the program is installed on both the Macintosh and the Newton, it takes 4 simple steps to use Bee-one!

- Database transfer, Set-up, Use, and Synchronization

(SBEEONE) Our Price **$139**

Web Ware

by BeachWare, Inc.

The ultimate collection of clip media and templates for building your own Web Page. An incredible selection of Shockwave movies, animated GIFs, buttons, bullets, dividers, and sample HTML pages. There are literally thousands of graphical elements on this disc, all there to spice up your web page. In all, it's about 300 megabytes of creativity only a mouse-click away! System Requirements: PC - 486 or better with 8 MB RAM, Sound card, SuperVGA, CD-ROM drive. Macintosh - Color Mac with 8 MB RAM, CD-ROM drive.

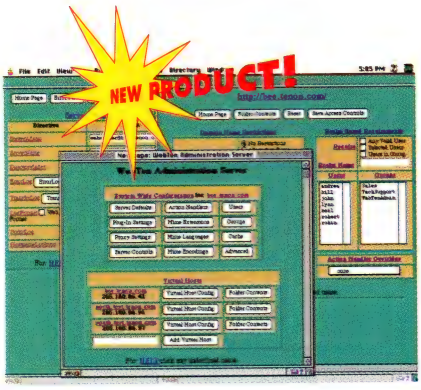
(SWEBW) Our Price **$24**

WAIT... There's More!

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
Fortran 77 SDK	SF77	699.00
ICONIX PowerTools-10 Pack	SICPP10	7,845.00
ICONIX PowerTools-6 Pack	SICPP6	5,945.00
ICONIX PowerTools-8 Pack	SICPP8	6,945.00
ICONIX PowerTools-AdaFlow	SICADA	1,395.00
ICONIX PowerTools-ASCII Bridge	SICASCII	1,395.00
ICONIX PowerTools-CoCoPro	SICCOCO	1,395.00
ICONIX PowerTools-DataModeler	SICDATAMOD	1,395.00
ICONIX PowerTools-FastTask	SICFASTTASK	1,395.00
ICONIX PowerTools-FreeFlow	SICFREEFL	1,395.00
ICONIX PowerTools-Object Modeler	SICOBJMOD	1,395.00
ICONIX PowerTools-PowerPDL	SICPOWER	1,395.00
ICONIX PowerTools-QuickChart	SICQUICKCH	1,395.00
ICONIX PowerTools-SmartChart	SICSMART	1,395.00
ICONIX Training & Consulting	TICONIX	2,945.00
IMSL Math and Stat Library	SIMSLSTAT	495.00
Info-Mac X	SINFOMAC10	39.95
Ionizer Real-Time Spectral Reshaping Tool	SIONIZER	800.00
LiveAccess™ 1 User Edition	SLAUE	69.00
LiveAccess™ 1 Developer Edition	SLADE	99.00
LiveCard	SLCARD	149.00
LJ Profiler	SLJPROF	295.00
MacFlow™: Flowchart Design and Development	SMACFLO	179.00
Mac Source II	SMACSOURCE	29.95
Nisus Writer 5.0	SNISUSW	220.00
Plan & Track™: Project Planning and Management	SPLNTRK	179.00
Screen Machine	SSM	24.00
Spyer	SSPY	39.00
Visual Café	SVCAFEMAC	199.00

WebTen
by Tenon Intersystems



WebTen is an industrial-strength, high-performance Apache Web server for Power Macs. WebTen's Web-based browser interface enables local or remote administration via your favorite browser. Since Apple's NeXT acquistion, Tenon has extended their unique "UNIX virtual machine" technology to produce a set of "Rhapsody-Ready" internet applications. WebTen is the first offering in this series.

- WebTen is the fastest Web server on Power Macintosh
- Sustains up to 10,000 connections a minute, or over 10 million connections a day
- Apache runs in Tenon's multi-threaded, pre-emptive multitasking environment
- Tenon's unique technology supports the widely acclaimed Apache Web server as a double-clickable Macintosh application

(SWEBTEN) Our Price **$495**

OOFILE Reporter Writer
by A.D. Software

- Full embedded report-writer, allows you to preview page-by-page and either print or save as plain text, HTML or RTF
- Multiple levels of breaks, database views, headers and footers are provided using a clean object-oriented design
- Incudes RAM-based version of OOFILE database. Included in full OOFILE Platform Bundle
- Saving to file without preview of printing is cross-platform-run on your Mac/Win/Unix server and creates web pages
- Price includes 1-year subscription

(SOORW) Our Price **$499**

WebSiphon
by Purity Software, Inc.

WebSiphon is one of the most anticipated new CGI products for Macintosh Webmasters delivering a complete authoring tool for truly revolutionary sites. This product alone can replace most all CGIs on your site resulting in increased dynamic serving speed, reliability, and powerful scripting and database abilities directly within your HTML pages! Also includes Verona, the fastest flat-file database server available for Macintosh web sites.

(SWSIPHON) Our Price **$495**

PageCharmer: Sizzling Effects...

PageCharmer 1.0
by Mainstay

PageCharmer is a set of customizable interactive applets that enhance web pages without writing a single line of HTML code. Whether the web site is already up and running or designing one from scratch, PageCharmer gives you the power to make it stand out from the crowd with sophisticated applets that can be personalized to fit most any need.

FEATURES:
LiveG-Map, LiveT-Map, LiveG-Button, LiveT-Button, LiveGT-Button, LiveG-Ticker, LiveT-Ticker, LiveG-Marquee, and LiveT-Marquee.

(SPGCHRM) Our Price **$139**

BBEdit 4.5
by Bare Bones Software

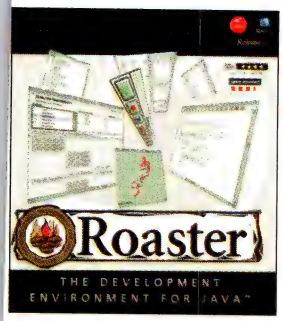
BBEdit 4.5 is a powerful, easy-to-learn text and HTML editor that offers developers and HTML authors the ability to build on its core functionality to suite their specific needs through its plug-in architecture and scripting capabilites. This new version includes: a visual table tool that speeds page and site development, contextual menu support for Mac OS 8, improved storage for 'grep' patterns, scriptable HTML authoring preferences and more. It still provides: unparalleled searching muscle with support for both 'grep' style and advanced literal searches, the ability to quickly compare differences between files or entire folders, integrated support for Symantec's IDE, Metrowerks CodeWarrior, THINK Reference 2.x, MPW Toolserver and most other environments and a heck of a lot more.

(SBBEDIT) Our Price **$119**

ObjectSet Mail SDK
by Smartcode Software

- Powerful C++ classes for integrating Internet e-mail in your applications
- Helps you write software that can share mail with other leading e-mail products
- Royalty-free MIME, SMTP, and POP3 APIs for Macintosh, Windows, and Unix
- Gives you the most robust MIME parser and encoder available
- Ideal for use in Internet and Intranet environments
- Comes complete with samples with documented, reusable source code
- Free standard technical support

(SOSMSDK) Our Price **$495**



Order Toll-free
800-MACDEV-1
(800-622-3381)

Roaster
by Roaster Technologies, Inc.

Get the most out of Sun's Java™ programming language with this powerful development environment.

- Features include: ability to build stand-alone Macintosh applications or applets; visual interface builder; ability to create cross-platform zip files; powerful Java debugger; wizard for quickly creating Java applets or applications; JDBC included; Java object database included; ability to call AppleScripts from Java; Just-in-time (JIT) compiler; JDK 1.0.2 support; class tree and hierarchical class browser; much more!
- Software includes: Over 300 example applets and applications; Netscape Internet Foundation Classes; Object Design's PSE for Java; OpenLink Software's JDBC Drivers; OpenSpace Java Generic Library; Microline Component Toolkit Lite 3.0; much more!
- Requirements: Runs on 68k or PowerPC, CD-ROM
- Price includes all web-based updates

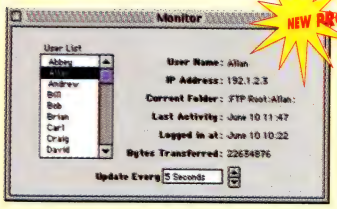
(SROAST3) Our Price **$99**

Power MachTen 4.0.3
by Tenon Intersystems

MachTen is the only Macintosh product that can turn your Macintosh into a complete Unix workstation. Based on BSD4.4 and the Mach kernel, MachTen brings the power of Unix to your desktop at an extremely attractive price point. MachTen enables you to:

- Run a high speed internet server, complete with WWW, FTP, NFS, DNS and print service
- Build a Mutihomed Web Server
- Develop applications in a Unix development environment, replete with the acclaimed GNU development toolset
- Program in Ada, C, C++, Pascal, Fortran, and more
- Run Xwindows applications, from remote workstations or on your Macintosh
- Run hundreds of Unix applications, already ported for MachTen and available on our Ported Applications CD-ROM
- Run Software.com Inc's acclaimed Post.Office mail transport service

(SM10PPC) Our Price **$695**



NEW PRODUCT!

Rumpus
by Maxum Development

Maxum's new, high-performance FTP server for the MacOS. Based on Maxum's RushHour TCP/IP implementation, Rumpus 1.0.1 offers the performance and reliability of high-end workstations with the ease of use, security, and flexibility of the Macintosh.

- Simplified setup, with no need to configure AppleShare, File Sharing, or Users & Groups for simple anonymous FTP
- Anonymous and/or secure server access, with separate security settings for anonymous vs. secure users
- Automatic MacBinary and Binhex encoding
- Complete logging, with separate anonymous and secure access logs, including anonymous user passwords
- Up to 32 simultaneous connections

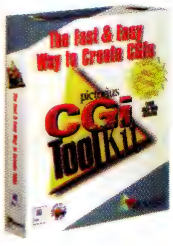
(SRUMP) Our Price **$195**

ScriptDemon
by Royal Software, Inc.

ScriptDemon is a browser plug-in that, for the first time, allows you to deliver and run AppleScripts from Web pages. The ScriptDemon plug-in will execute the embedded AppleScript code included on a Web page. ScriptDemon painlessly and inexpensively handles many previously impossible tasks, such as:

- Using the Intranet to manage all Macintosh computers on-line
- Using the Internet to install and configure software
- Using the Internet to configure hardware
- Delivering complex sets of files and assembling them on the browsing computer
- Providing interactive education and product support on both the Internet and Intranet
- A perfect companion to LiveCard!

(SSDEMON) Our Price **$949**

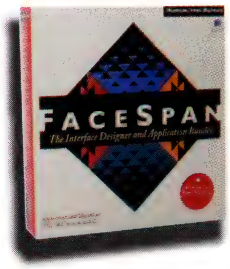


CGi Toolkit
by Pictorius Inc.

The Pictorius CGi Toolkit is the fast and easy route to high performance CGIs and ACGIs for your Mac Web site.

- Interactively develop CGIs while the web server, the CGI Toolkit and the browser are running on the same machine
- Interactively develop, test and debug CGIs before compiling
- Powerful debugger allows you to edit code, roll back, code and change input values while your application is running
- Fully object oriented so you can re-use your code
- Automatic handling of Apple Events so you can concentrate on building functionality
- Easy creation of multi-function CGIs which reduces application footprint and RAM usage

(SCGITLKT) Our Price **$149**



FaceSpan v2.1

by Digital Technology International

- Develop integrated software, make stand alone applications, create friendly interfaces
- Develop quick prototypes, print multiple pages with sophisticated layouts
- Script essential elements of the FaceSpan application - Enhanced save options
- Play and record sounds as either "snd" resources or as "AIFF" files
- Create miniature or complete apps that run on either Power PC or 68k computers
- Use precise time measurement for implementing timed behaviors — New properties
- Proportionally scale PICT images -Align images in a pictbox - Automate any application
- Monitor and respond to low-memory situations-Increased support for Frontier UserTalk!

(SFACESPAN) Our Price **$299**

PreFab Player

by PreFab Software, Inc.

PreFab Player is a faceless background application (similar to a system extension) that lets your scripts query and control otherwise non-scriptable applications, desk accessories and control panels. Player adds verbs that directly manipulate the Macintosh user interface: choose from menus & pop-ups, select radio buttons, type text, determine the name of the frontmost window, the state of a check box, etc. Balloon help identifies non-standard dialog items.

- Adds verbs to AppleScript and to Frontiers UserTalk
- Controls the Frontmost Application
- Balloons Identify User Interface Objects

(SPLAYER) Our Price **$95**

Scripter 2.0

by Main Event Software

For professionals, for novices, for webmasters, for solutions providers, there's only one serious choice. Scripter!

- Scripter and FaceSpan work together: one click opens your FaceSpan script in Scripter, another sends it back
- Debug handlers without modifying your scripts using the Call Box
- Applet simulation, live editing, Object map, associated terminology
- Search backwards, block generators, more navigation shortcuts, more drad-and-drop, and an even more enhanced trace log
- Now Includes ScriptBase; stores your data and media elements and share them between scripts all with a special new browser
- Easily write and compile scripts that have handler declarations and other vocabulary specific to a particular scriptable application
- Scripter is the natural companion to AppleScript for users at all levels of proficiency. Don't write scripts without it!

(SSCRIPTER) Our Price **$199**

AppleScript Software Development Toolkit 1.1

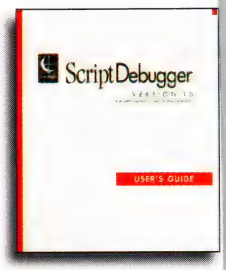
by Apple Computer, Inc.

- AppleScript language, system software extension, and script editor
- FaceSpan 1.0
- Developer's redistribution license for AppleScript System software extension and FaceSpan runtime code

(SASDT) Our Price **$49**

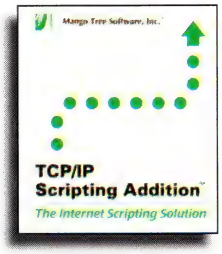
Script Debugger

by Late Night Software Ltd.



- A powerful and flexible AppleScript authoring tool – get the most from AppleScript!
- Advanced debugging environment offers single-step script execution with breakpoints
- Script Debugger dictionary browser features a graphical view of objects provided by scriptable applications
- Includes Late Night Software Scripting Additions – a collection of more than 70 new AppleScript commands, and Scheduler, a utility that allows you to launch scripts at pre-determined times

(SDEBUG) Our Price **$129**



TCP/IP Scripting Addition

by Mango Tree Software

- Award-winning AppleScript scripting addition
- Allows you to write scripts using MacTCP™ commands in AppleScript™
- Send e-mail or files through a script, check if users are logged on (via Finger), automate FTP, Gopher, NetNews, Telnet, and LPR, verify links in HTML documents, and quickly write many other TCP/IP client-server programs
- Works with AppleScript, MacTCP 2.0.4 and Open Transport

(STCP) Our Price **$49**

NEW PRODUCT!

WindowScript

by Royal Software, Inc.

WindowScript is the ultimate tool for designing Macintosh user interfaces using HyperCard. Design Real "Macintosh" user-interfaces right inside HyperCard. Until now you either created HyperCard stacks or Macintosh applications. With WindowScript you can literally bring the look and feel of a real Macintosh user-interface to HyperCard. If you're a HyperCard developer, interface designer, application developer, program manager or tester searching for a prototyping tool, WindowScript is perfect for the job.

(SWSCRIPT) Our Price **$149**

Apple Media Tool Programming Environment 2.1

by Apple Computer, Inc.

This object-oriented language and application framework allows programmers to customize features used within the Apple Media Tool authoring environment

- Includes an expanded Apple Media Language (AML) class library, incremental compiling and linking of AML code, faster debugging facilities, Macintosh Programmers' Workshop (MPW), and user-oriented documentation written from an AMTPE developer's perspective
- Portable across 68K, Power Macintosh, and Windows platforms

(SAMTPE) Our Price **$995**

Multimedia Authoring with Apple Media Tool

by Apple Computer, Inc.

Apple Media Tool offers new multimedia users a way to get started creating interactive multimedia with minimal learning time. This self-paced tutorial will make Apple Media Tool (AMT) even easier to understand and to use. Using this tutorial, you will create a realistic multimedia project using exciting techniques such as QuickTime movies, animation and more. A demo version of AMT is included and can be used for the exercises. Training Format: Tutorial with labs.

(SMWAMT) Our Price **$49.95**

Virtual Reality Programming with QuickTime VR 2.0

by Apple Computer, Inc.

- Virtual Reality Programming Book/CD-ROM for QuickTime VR 2.0
- Enables you to write C and C++ programs using QuickTime VR 2.0
- Allows QuickTime VR to be used in games, multimedia titles and other programs
- QuickTime VR 2.0 objects can be zoomed in on, panned, or linked with hots spots
- Both panoramas and objects have hot spots linked to World Wide Web URLs

(SVRPQT) Our Price **$49**

QuickTime Developer's Kit 2.0

by Apple Computer, Inc.

- QuickTime 2.0 Extension, QuickTime Power Macintosh Extension, and QuickTime Musical Instruments extension
- Utilities like MoviePlayer 2.0, 16-bit Audio Compression, etc.
- Sample content such as MPEG Movies, Music Movies, Time-Code Movies, and 60 field per second movies
- Includes software-only playback features such as faster 2x playback mode for current compressors, Apple Cinepak compressor, 1-bit fast dithering, network tuning, load-into-RAM option, and Photo CD support

(SQTDK) Our Price **$99**

Clip VR™

by eVox Productions

Clip VR™ is a new digital image library offering high quality Photographic Virtual Reality (PVR) images for use with Quicktime® VR and other desktop VR tools.

Clip VR™ Panoramic Image components include alpha channel masks. Combine elements into a composite panorama which is converted to the finished QuickTime VR movie using Make QTVR Panorama Tool. The Components ("Clips") include complete 360 degree scenes, libraries of 360 degree terrains, 360 degree skies, buildings, and objects. Images are provided as .PICT files AND QuickTime VR movie files. Clip VR™ allows you to create high quality VR worlds from pre-photographed component images. Clip VR™ can be used to add excitement to a web site, to increase the interactive value of a CD-ROM, or simply for fun! Requires imaging editing program such as Adobe Photoshop™ to perform .PICT image compositing.

(SCLIPVR) Our Price **$89.95**

QuickTime VR 2.0 Authoring Tools Suite

by Apple Computer, Inc.

- QuickTime VR is a cross-platform software from Apple which enables webpage designers and professional developers to create new multimedia products and webpages incorporating QuickTime VR content. With QuickTime VR, users interactively navigate through 360° views of space, and explore three dimensional objects on Macintosh or Windows-based personal computers
- The QuickTime VR Authoring Tools Suite is a set of Macintosh tools to create and link panoramas and objects from photographic, digital, video, or computer generated images
- Included is a complete set of documentation for planning, designing, photographing, and creating QuickTime VR panoramas and objects. The authoring tools also allow you to link objects to panoramas using clickable hot spots

Included on the CDs are:

- A software tool (MPW-based) that stitches and blends adjacent images into a panoramic PICT file
- A software tool (MPW-based) that dices and compresses panoramic PICT files to less than 100 KB (low resolution) per panorama
- A scene editor (HyperCard-based) to create QuickTime VR scenes by adding and positioning nodes, hot spots, linking nodes together, and for linking QuickTime VR objects to scenes
- A variety of utility tools for formatting the data into the runtime software

Due to a revolutionary distortion-correcting algorithm, QuickTime VR panoramas and objects maintain a normal perspective when the user moves the mouse. The speed of the algorithm allows up to 24-bit color images. Both vertical and horizontal panning can occur at fast speeds.

(SQTVRATS) Our Price **$395**

Terran Interactive

NEW PRODUCT!

Media Cleaner Pro
by Terran Interactive

Use Media Cleaner Pro 2.0 to optimize and compress video for CD-ROM, kiosk, or the Internet. Media Cleaner Pro automates your work flow allowing you to get the highest quality video, faster and easier than any other program on the market.

- Includes Adobe Premiere Export module
- Optimal palette generation, Drag-and-drop batch processing
- RealMedia, VDOLive and improved QuickTime support
- Dynamic Preview Window, the Media Wizard, multiprocessor support and more!

System Requirements:
68040 Mac or better (PowerPC strongly recommended, req'd for RealMedia), QuickTime 2.0 or later (2.5 strongly recommended) 8 Mb application RAM, MacOS 7.0.1 (7.5 or later recommended) SoundManager 3.2, CD-ROM Drive

(SMCP) Our Price **$359**

Registered owners of Movie Cleaner Pro 1.3 or earlier can upgrade

(SMCPUP) Our Price **$129**

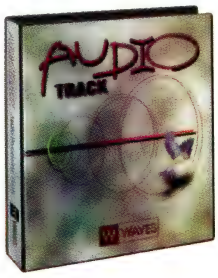
Captivate 4.6: Essential Graphics Utilitie
by Mainstay

Captivate™ 4.6 is a powerful collection o graphics utilities for Macintosh, based on Mainstay's acclaimed screen capture utilit graphics and multimedia scrapbook, and graphics viewer. Captivate provides essential graphics utilities to th professional and hobbyist alike.

- Package includes: Captivate Select, Captivate View, and Captivate Sto
- Any one of the three can be used alone, and together they make an unbeatable team
- Whether writing a training manual, creating an ad, or just creating a startup screen from your favorite picture, Captivate is everything professionals need at a price anyone can afford.

(SCAPTIV) Our Price **$79**

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.



AudioTrack
by WAVES

AudioTrack is a software plug-in for native processing o digital audio recording and editin systems. Waves AudioTrack combines the most-needed audi processors into a single piece of software, including 4 bands of equalization, compression/expansion, and noise gating. AudioTrack is ideal for preparing audio for InterNet streaming formats, processing individual mono/stereo tracks of audio and audio for video editing systems including digital sequencers.

Feature list

- A single window interface
- 4-band ParaGraphic Equalizer, Compressor/Expander and Gate
- Instantaneous A/B comparisons of on-line settings
- Pretested setup libraries supplied for various processing solutions
- Power Macintosh native processing (requiring no DSP board)
- Volume and gain reduction meters
- Peak hold and clip meters

Requirements:
The AudioTrack is compatible with all machines supported by Deck II, SoundEdit 16 Adobe Premiere 4.0 and Cubase VST 3.1

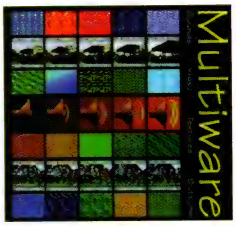
(SAUDIOTRK) Our Price **$270**

Music Tracks
by BeachWare, Inc.

A new PC/Mac & Audio multimedia music CD-ROM. The clips include musical introductions, fanfares, background music, and more. This collection offers you 100 music clips stored in .WAV format for Windows, SoundEdit & AIFF formats for Macintosh and as Audio tracks for audio CS players. All of the music clips are completely license and royalty-free!! Mac System requirements: Mac Plus or greater, CD-ROM drive. PC system requirements: Windows 3.1 or later, Sound Blaster compatible board, CD-ROM drive.

(SMT) Our Price **$24**

Order Toll-free
800-MACDEV-1
(800-622-3381)



MultiWare Multimedia Collection
by BeachWare, Inc.

Introducing a new Audio multimedia music CD-ROM for the Macintosh. This disc is a collection of clips ideal for Desktop Presentations and other Multimedia applications. This incredible collection of license-free media clips is bursting with 240+ color pictures and backdrops (PICT), 200+ sound & music clips (SoundEdit), 140+ QuickTime movies, and a variety of multimedia tools for use with the Macintosh.

(SMWMC) Our Price **$24**

Developer DEPOT™

ORDER FORM

http://www.devdepot.com
E-mail: orders@devdepot.com
U.S./Canada: 800-MACDEV-1 (800-622-3381)
Outside U.S./Canada 805-494-9797
Fax: 805-494-9798

Please fill out as completely as possible to avoid delays in processing your order.

Name _____

Company _____

Address _____

City _____ State/Province _____

Zip/Postal Code _____ Country _____

Home Phone _____ Business Phone _____

E-Mail Address _____ Fax _____

☐ Cash ☐ Check/Money Order ☐ VISA ☐ MasterCard ☐ American Express Expiration Date _____

Card # _____ Signature _____

ORDERING INSTRUCTIONS: Itemize the products you want below. Fill in the Order Total, Sales/Use Tax (for California residents only), and Subtotal. Shipping & Handling charges will be added upon processing. All orders will be shipped via most economical method (e.g., UPS Ground), unless you state otherwise. **Remember to fill out your address and payment information above.**

PRODUCT NUMBER	ITEM DESCRIPTION	QUANTITY	UNIT PRICE	TOTAL

If you are ordering a subscription to **MacTech™ Magazine**, are you a new subscriber, or are you renewing your subscription?

☐ New Subscription
☐ Renewal

*Magazine subscriptions are non-taxable

Order Total _____

Sales/Use Tax (CA residents only)* _____

SUBTOTAL _____

FOR OFFICE USE ONLY

Shipping & Handling _____

TOTAL _____

Fold Here First

Fold Here (Last) and Tape Closed (not staple)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 264 THOUSAND OAKS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

XPLAIN CORPORATION
P O BOX 5200
WESTLAKE VILLAGE CA 91359-9864

Casino!

by BeachWare, Inc.

Can't make it to Vegas this month? Your best bet is Casino! Whether your favorite is Slots, Poker, Blackjack, or Keno, this virtual casino will entertain you for hours with its ten different machines. Mac System equirements: Color Mac, CD-ROM drive, 4 MB of RAM. PC system equirements: Windows 3.1 or later, CD-ROM drive, 4 MB of RAM.

(SCAS) Our Price **$24**

Abuse

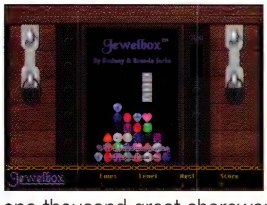
by Bungie Software

Abuse is 360 degrees of side-scrolling action. Run, jump, fall and fly in any direction - through industrial corridors, caverns and sewers. Destroy enemies in any direction with grenade launchers, rocket launchers, napalm and nova spheres! Avoid deadly traps with jet packs and turbo boost!

Key Features:

- Make your own mayhem with the Level Editor
- Hunt down your friends in 8-person multiplayer games
- Awesome Arsenal. Napalm Bombs, Nova Spheres and the Death Saber: just a few ways to lay waste!
- Point and Kill Interface. Move and annihilate mutants in complete 360° freedom
- Blast your way through floors, walls and ceilings in search of the ultimate power-up!
- Abuse is 360 degrees of side-scrolling action. Run, jump, fall and fly in any direction – through industrial corridors, caverns and sewers
- Destroy enemies in any direction with grenade launchers, rocket launchers, napalm and nova spheres! Avoid deadly traps with jet packs and turbo boost!

(SABUSE) Our Price **$51**



1000 Games for Macintosh

by BeachWare, Inc.

The best Macintosh game disc in the entire world, this CD-ROM contains over one thousand great shareware and public domain programs. Battle ugly aliens, blast apart run-away asteroids, deal yourself that royal flush or solve that 3-D puzzle, this disc has it all! System requirements: Mac Plus or greater, CD-ROM drive, and 2 MB of available RAM (4 MB of RAM when running under System 7).

(STGM) Our Price **$24**

Classic Arcade

by BeachWare, Inc.

Ten of your favorite coin-arcade games, redone with killer graphics and sounds! Walk through a virtual arcade and test your game playing skills with these exciting arcade classics. Ten games, including Moon Lander, Astro-Boing, Hyper Hockey, Ballistic Avenger, and more. System Requirements: Mac - Color Mac with 8 MB RAM, CD-ROM drive. PC 486 with 8 MB RAM, Sound card, SuperVGA,CD-ROM drive.

(SCLA) Our Price **$24**

Marathon Trilogy Box Set

by Bungie Software

The Marathon Trilogy Box Set brings all three Marathon games together in one affordable package, with tons of extras thrown in. Besides Marathon, Marathon 2: Durandal and Marathon Infinity, you'll also receive a staggering 1200 maps, featuring never-released Bungie maps and the winners of the Infinity Mapmaking Contest, The Marathon Scrapbook (a behind-the-scenes look at themaking of the Marathon games), Marathon collectables like the Marathon 3-sticker set, and to top it off, the award-winning game that laid the groundwork for Marathon: Bungie's breakthrough Pathways Into Darkness.

The Marathon Trilogy Box Set is native to the Power Macintosh, utilizes the graphics acceleration of 630 and 6200 machines, is 8, 16 and 24-bit color capable, and can be played with joysticks and game pads. The package requires a 68040 or higher Macintosh,CD-ROM drive, 8-bit color monitor (13" recommended), and System 7 or later.

(SMTBS) Our Price **$65**

Trivia Warehouse 2000

by BeachWare, Inc.

Introducing a fun new PC and Mac CD-ROM. This disc contains two thousand trivia questions, in 45 categories, in several game formats! Test your memory with the Q&A, Concentration, and Multiple Choice games! Categories include: Animals, Bodies, Bond, Bugs, Cities, Comics, Geography, Gilligan, Gross, Health, Holidays, Horrors, Kids, Knot's Landing, Math, Movies, and many more. Mac System requirements: Color Mac, CD-ROM drive, 4 MB of RAM. PC system requirements: Windows 3.1 or later, CD-ROM drive, 4 MB of RAM.

(STW2K) Our Price **$24**

WAIT... There's More!

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
A Zillion Sounds	SAZS	24.00
Night Sky Interactive	SNSI	24.00

System 7.5 Technologies

by Apple Computer, Inc.

- Self-paced course designed to allow software developers to write code that extends the functionality of an application for System 7.5
- Contains comprehensive materials for drag-and-drop, threads, standard mail package, and QuickDraw GX printing

Students should be familiar with the basics of developing an application on the Macintosh. Metrowerks CodeWarrior Lite is included on the CD with the lab exercises. The lab assignments were developed in CodeWarrior 8. The labs can also be done in another development environment but project files for them are not provided.

Training Format: Tutorial with labs.

Requirements: Macintosh or Mac-OS compatible computer with a 68020 processor or greater (PowerPC preferred); 8 MB RAM; 25 MB hard disk space; System 7.5 or later; CD-ROM drive.

(SSYSTECH) Our Price **$49.95**

Apple Guide Integration

by Apple Computer, Inc.

- Self-paced overview teaches you when an how to add Apple Guide help to your prograr
- Powerful help system that can guide the user through a task.
- Tutorial will lead you through the steps necessary to integrate Apple Guide into your application. CodeWarrior Lite is included or the CD with the lab exercises.

Training Format: Overview with labs.

Requirements: Macintosh or Mac-OS compatible computer with 68020 processor or greater, PowerPC preferred; 8 MB RAM; 25 ME hard disk; System 7.5 or later; CD ROM drive.

(SAGI) Our Price **$49.95**

Virtual Tutor for QuickTime VR

by Apple Computer, Inc.

- Self-paced, hands-on course, which provides a comprehensive environment for learning the steps of the QTVR development process. The student can cover all of the topics or choose areas to focus on. Topics covered include: QTVR capabilities and key concepts, panoramic movies, object movies, QTVR Scene movies and authoring with QTVR
- CD-ROM contains lots of useful examples and demos. In addition to all the step-by-step exercise files

If the student completes the entire course, he/she will create a complete, authored multimedia project similar to the demonstration title that comes on the enclosed CD-ROM. There are approximately 3–4 days of training.

Training Format: Tutorial with labs.

Requirements: 40 MB RAM minimum, 64 MB preferred; Macintosh or Mac OS-compatible computer with a 33 MHz 68040 processoror greater; System 7.1 or later; CD-ROM drive; 17" color monitor.

(SVTFQTVR) Our Price **$79.95**

develop

Apple's develop™ back issues are available for only $10 per copy. To place your orders, call 800/MACDEV-1.

For Macintosh Programmers & Developers
MacTech® MAGAZINE

MacTech® Magazine

MacTech keeps Mac programmers & developers up to date with everything they need to know about software development. Topics like Rhapsody, Java, QuickTime, OPENSTEP, Objective-C, C/C++, Object Oriented Technologies, product reviews and much more!

Subscriptions:

(MTYRDM) US/Domestic for 12 issues **$47**

(MTYRCM) Canadian for 12 issues **$59**

(MTYRFM) International for 12 issues **$97**

Back Issues: each plus shipping (subject to availability) **$10**

metrowerks

GeekWare™

You live it, you breath it... you might as well wear it! (XL only)

Alien T-Shirt
(AALIEN) Our Price **$9.95**

Arnold T-Shirt
(ACWARNLD) Our Price **$9.95**

Arnold Jr. T-Shirt
(AARNOLDJR) Our Price **$9.95**

Blood, Sweat & Code T-Shirt (SS)
(ACWSBLOOD) Our Price **$9.95**

Blood, Sweat & Code T-Shirt (LS)
(ACWLBLOOD) Our Price **$14.95**

CodeWarrior Baseball Cap - Black
(ACWBHAT) Our Price **$14.95**

CodeWarrior Sweatshirt - Black
(ACWSWEAT) Our Price **$29.95**

CodeWarrior Hawaii Five-0 Shirt
(ACWHAWAII) Our Price **$9.95**

CodeWarrior Winter Hat
(AWINHAT) Our Price **$14.95**

Debugger Boxer Shorts
(ADBOXER) Our Price **$16.95**

Discover Programming T-Shirt
(ADISCPT) Our Price **$9.95**

LEGTOP PODEUM

Podeum Sport

by Rach, Inc.

Now you can have laptop stability, drop protection & mobility. If you're looking for an inexpensive, non-technical gift for a laptop owner – look no further.

- Strap your laptop to your leg
- Universal size - fits all laptops and all legs (13" - 46")
- Velcro Velcoins attach your laptop to the podeum
- Podeum allows working angles up to 90 degrees
- Dropped laptops account for 41% of all laptop fatalities
- Manufacturer's lifetime warranty.

(APODSP) Our Price **$39**

MacTech® Mouse Pad

Slide on this! With an extra-large surface (11" by 10") and a deluxe sleek plastic coating, you'll be zooming across your screen in no time at all. Speed limit not enforced!

(AMTPAD) Our Price **$8.95**

APPLE ENTERPRISE SOFTWARE

Getting Started With WebObjects
by Apple Enterprise Software

If you're a first time user, start here to learn the basics of how to create and run WebObjects applications.
(BGSWO) Our Price **$14**

WebObjects Developer's Guide
by Apple Enterprise Software

A guide to building and understanding WebObjects applications. Takes a close look at the WebObjects scripting language. Additional sections explain the WebObjects architecture and tells you how to integrate your code into the request-response loop, create reusable components, create client-side components, take advantage of powerful Foundation Framework features, and more. Filled with example code. For all WebObjects programmers.
(BWODG) Our Price **$16**

D'OLE Developer's Guide
by Apple Enterprise Software

Distributed OLE is available today, and this book shows you how to use it. Using real-world examples, the book steps you through the process of making your OpenStep objects available on Windows through OLE.
(BDOLEDG) Our Price **$22**

Discovering OPENSTEP, Mach
by Apple Enterprise Software

Introduces programmers to NeXT's OPENSTEP 4.0 Developer product by guiding them through the creation of three applications of increasing complexity. The tutorials demonstrate and explain programming techniques, Objective-C fundamentals, common APIs, and usage of the developement tools. Along the way they present summaries of important concepts and paradigms. The book also includes a chapter directing readers to programming resources, further information, and services such as training and support. An appendix offers a concise discussion of object-oriented programming.
(BDOSTEPM) Our Price **$15**

Discovering OPENSTEP, Windows
by Apple Enterprise Software

Discovering OPENSTEP provides an introduction to OPENSTEP programming on Wlndows NT. It guides the reader through the creation of three applications of increasing complexity. Along the way, it explains concepts and illustrates aspects of Objective-C, OpenStep classes, the development environment, and programming techniques. A short appendix offers a summary of object-oriented programming.
(BDOSTEPW) Our Price **$16**

Object-Oriented Programming and Objective C
by Apple Enterprise Software.

An introduction to the principles of object-oriented programming in OPENSTEP and the official description of the Objective-C language. Objective-C is easy to learn and use because it adds very little syntax to the C programming language. It's dynamic nature allows you to accomplish things not possible in most other object-oriented languages. For any OPENSTEP programmer.
(BOOPOC) Our Price **$24**

Working w/ Interface Builder (for EOF)
by Apple Enterprise Software

A hands-on, award-winning book designed to help you get your job done with the updated Interface Builder, released with NEXTSTEP 3.3 and the Enterprise Objects Framework 1.1. For any programmer using Interface Builder to design objects that truly work in NEXTSTEP.
(BWIB) Our Price **$24**

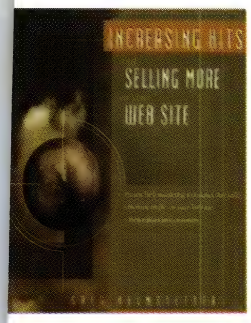
Using EOF 2.1 w/ OPENSTEP (Mach & Windows)
by Apple Enterprise Software

Using Enterprise Objects Framework with OPENSTEP describes how to create an Enterprise Objects Framework application on OPENSTEP. It includes a tutorial and a chapter on creating a user interface for an OPENSTEP Enterprise Objects Framework application.
(BUEOFO) Our Price **$14**

EOF Developer's Guide for EOF 2.1 (Mach & Windows)
by Apple Enterprise Software

The Enterprise Objects Framework Developer's Guide describes how to develop database applications using the Enterprise Objects Framework tools and classes. It includes an architectural overview of the product, and descriptions of programming tips and techniques. An appendix offers a summary of Entity-Relationship Modeling.
(BEOFDG) Our Price **$24**

EOF Developer's Guide for EOF 2.0 (BEOFDG20) Our Price **$24**
EOF Developer's Guide for EOF 1.x (BEOFDG1X) Our Price **$24**

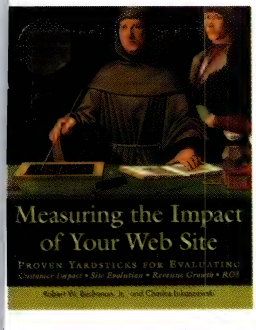


Increasing Hits and Selling More on your Web Site

by Greg Helmstetter

Written especially for entrepreneurs, corporate marketing managers, small business owners, and consultants, this valuable guide gives you rare tips and tricks you need to know to make your site a commercial success.

(BIHSMWS) Our Price **$22.45**

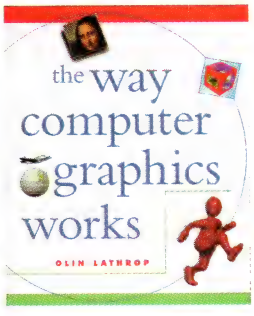


Measuring the Impact of Your Web Site

by Robert W. Buchanan and Charles Lukaszewski

This book features case studies from many successful and some less successful corporate web site pioneers. You will learn techniques and commercially available tools for measuring site traffic and visitor behavior–and help you choose the right ones for the job. Written by leading corporate site consultants this book tells you how to develop a management strategy geared toward optimizing web site productivity.

(BMIYWS) Our Price **$26.95**



The Way Computer Graphics Works

by Olin Lathrop

A complete guide to mastering computer graphic basics. It is written in a frank, down-to-earth style covering everything from how computer graphics are different from fine art and photographs, to modeling, pixels, and the principles of animation. All of this is done without resorting to mind-numbing equations and impenetrable technical jargon.

(BWCGW) Our Price **$29.65**

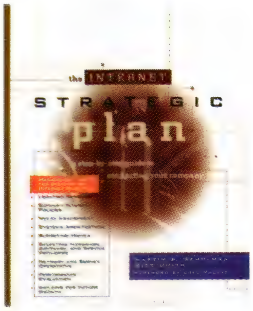
Debugging Macintosh Software with MacsBug

Includes CD-ROM!

by Konstantin Othmer and Jim Straus

MacsBug, from Apple Computer, Inc., is the leading debugging software program for the Macintosh. This book/disk package is an all-in-one kit for using MacsBug. Chapter 1 introduces MacsBug and describes the contents of the rest of the book. Chapter 2 describes how to install MacsBug and enough low level details about the Macintosh so that you can use MacsBug. Includes MacsBug 6.2 on disk.

(BDMSWM) Our Price **$31.45**

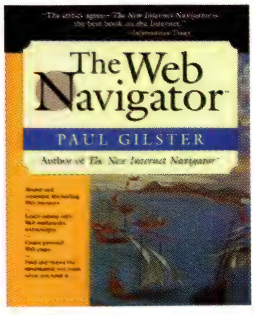


The Internet Strategic Plan

by Martin A. Schulman and Rick Smith

This book gives you all of the management tools you need for creating a strong Internet and Web presence. A blueprint for your strategic plan, this book guides you every step of the way in establishing your company's place on the Internet and World Wide Web.

(BTISP) Our Price **$22.45**

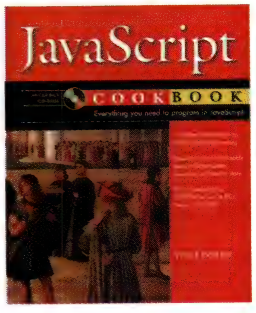


The Web Navigator

by Paul Gilster

This book keeps you right on top of all the recent changes in the Web, tells you what's out there right now and what's coming in the future. You'll get samples of various Internet sites and candid discussions of providers. You'll receive proven strategies for finding and managing information.

(BTWN) Our Price **$22.45**



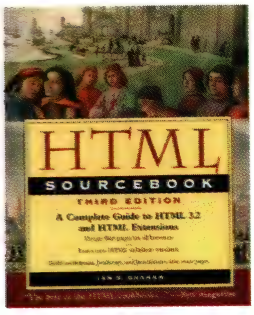
JavaScript Cookbook

Includes CD-ROM!

by Yosef Cohen

Everything you need to master the fundamentals of JavaScript programming is in this book/CD-ROM package. A special easy-to-use reference section offers detailed analysis and examples of objects, properties, event handlers, and statements. You'll also find all the latest information relevant to JavaScript programming since the advent of Navigator 2.0.

(BJSCB) Our Price **$44.99**



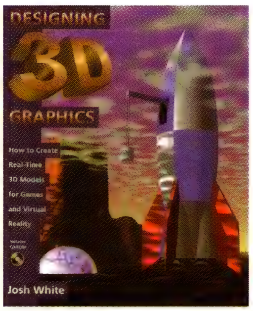
HTML Sourcebook, 3rd Edition

by Ian S. Graham

Critics everywhere agree, HTML Sourcebook is the best guide to HTML for Web professionals. That's because no other book makes it so easy for you to quickly master all the commands, tools, and expert techniques you need to create cutting-edge Web page documents. Completely revised and expanded by nearly 50 percent, this new third edition of the best-selling guide to HTML gives you the complete lowdown on all the changes and enhancements to the HTML, HTTP, and URL standards.

(BHTMLS) Our Price **$26.95**

Check out our Web site!
• Full product descriptions • Hundreds of more products
http://www.devdepot.com



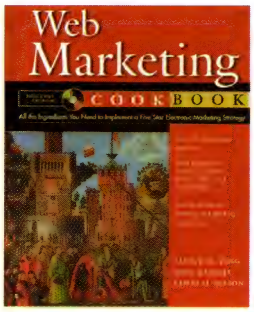
Designing 3D Graphics

by Josh White

Includes CD-ROM!

In this powerful book/CD-ROM package, top computer graphics artist Josh White tells you everything you need to know to create sophisticated real-time 3D graphics for computer games and virtual reality. This book contains the in-depth knowledge of software tools and hands-on modeling techniques that Josh White has learned while creating artwork for over 20 commercial games, including Descent, Zone Raiders, Locus, Legoland, and others.

(BD3DG) Our Price **$35.95**



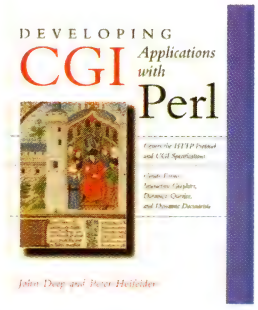
Web Marketing Cookbook

by Janice M. King, Paul Knight, and James H. Mason

Includes CD-ROM!

Create the ultimate Web marketing site, quickly and painlessly! Learn how to build a Web site for your small business or nonprofit organization with this step-by-step approach. This book/CD-ROM package contains your simple, non-technical tools for converting your print brochures, text, and graphics into a powerful online promotion.

(BWMCB) Our Price **$35.95**



Developing CGI Applications with Perl

by John Deep and Peter Holfelder

A complete step-by-step guide to creating sophisticated interactive CGI applications using Perl. If you're ready to build your own customized interactive documents, forms, graphics, and other full-feature CGI applications using Perl, then this book will show you how. Covers CGI, HTTP, and the Perl scripting language.

(BDCGIA) Our Price **$26.95**

Rhapsody Developer's Guide

by Jesse Feiler

Covers the basic architectural principles of Rhapsody: the Mach microkernel, object-oriented programming, and the elements of a modern OS such as preemptive multitasking, protected memory, and symmetric multiprocessing. Also shows ways of getting to this new environment—Objective C, conversion tools, and the integration of Java—to develop Rhapsody products. Paperback, 450 pages.

(BRDG) Our Price **$35.95**

Practical Object-Oriented Development in C++ and Java

by Cay S. Horstmann

This book offers advice on real-world ways to use these powerful programming languages and techniques. Using the Unified Modeling Language (UML) methodology, expert Cay S. Horstmann gives you clear, concise explanations of object-oriented design, C++, and Java in a way that makes these potentially daunting operations more accessible than they've ever been before.

(BPOOD) Our Price **$31.50**

WebMaster in a Nutshell, Deluxe Edition

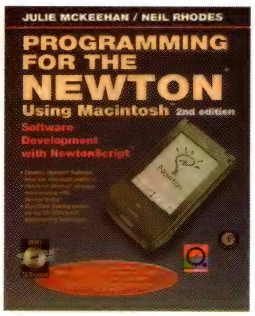
by O'Reilly & Associates, Inc.

Includes CD-ROM!

Cross-platform, completely portable, and lightning fast, the CD-ROM is an invaluable addition to the webmaster's toolbox. The CD-ROM contains the Web Developer's Libaray—the full text of the latest editions of five popular O'Reilly titles: "HTML: The Definitive Guide, 2nd Edition"; "JavaScript: The Definitive Guide, 2nd Edition"; "CGI Programming on the World Wide Web"; "Programming Perl, 2nd Edition"; and "WebMaster in a Nutshell." The Deluxe Edition also includes a printed copy of "WebMaster in a Nutshell," the all-inclusive quick reference that belongs next to every webmaster's terminal. Includes CD-ROM & 356 page book.

Requirements: The CD-ROM is readable on all platforms, but requires a web browser that supports HTML 3.2, Java, and JavaScript.

(BWMNUTD) Our Price **$62.95**



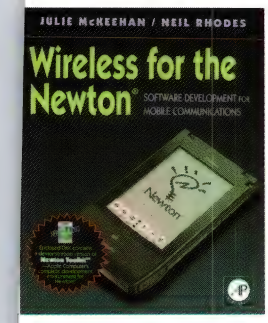
Programming For The Newton Using Macintosh, 2nd Edition

by Julie McKeehan and Neil Rhodes

Includes CD-ROM!

This book gives you everything you need to create Newton 2.0 applications. From the people who developed the Newton programming training materials for Apple Computer, this book uses a clear and comprehensive approach to teach you how to program the Newton. Includes a CD-ROM full of sample code and additional goodies. The samples include solutions to all of the coding examples found in this book. Each example and exercise also has a narrated QuickTime movie showing the solution from start to finish in Newton Toolkit.

(BPFNUM2) Our Price **$31.45**



Wireless For The Newton

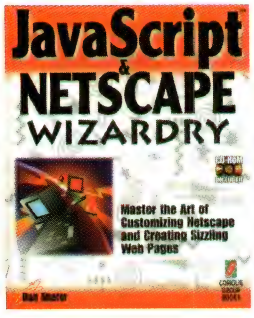
Includes Disk!

by Julie McKeehan and Neil Rhodes

A book that picks up where Programming for the Newton left off, teaching the reader how to develop Newton software on the Macintosh. The enclosed floppy disk provides a sample application, as well as a fully functional demonstration version of Newton Toolkit.

- Learn to develop Newton software on the Macintosh
- Hands-on Newton environment training with sample code
- Includes disk with sample source code for a Newton application, as well as demonstration NTK – the complete development environment for the Newton

(BWIRELESS) Our Price **$31.45**



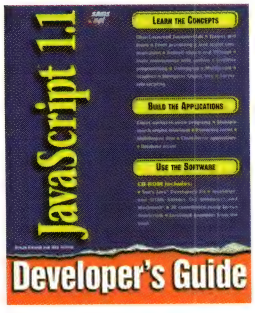
JavaScript & Netscape Wizardry

by Dan Shafer

The perfect book to show you how to turn Netscape into your own personal, customized operating system. Provides the inside tips and techniques for making your Web pages much more attractive. Shows you how to use all of the key features of the JavaScript language, including objects, methods, properties, events, and much more. Includes CD-ROM with numerous interactive scripts written in JavaScript you can add to your Web pages today. A complete set of the best Java applets. Useful plug-ins designed to supercharge Netscape and resources to help JavaScript programmers.

Includes CD-ROM!

(BJNWIZ) Our Price **$31.45**



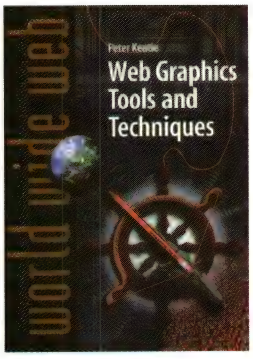
JavaScript 1.1 Developer's Guide

by Arman Danesh and Wes Tatters

Written by developers for developers. An advanced guide to creating professional Web applications with JavaScript 1.1 as deployed in Netscape Navigator 3.0, Microsoft Internet Explorer 3.0, and LiveWire. Includes CD-ROM with Sun's Java Developer's Kit, JavaScript and HTML Editors for Windows and Macintosh, 20 contributed ready-to-run JavaScripts and JavaScript examples from the book.

Includes CD-ROM!

(BJSDG) Our Price **$44.99**



Web Graphics Tools and Techniques

by Peter Kentie

The ultimate source of information on Web graphics for both Macintosh and PC users. Recent technologies covered include: ActiveX, Sound, Adobe Acrobat, Java, VRML, QuickTime VR and video, Shockwave, and 3D Animation.

(BWGTT) Our Price **$35.95**

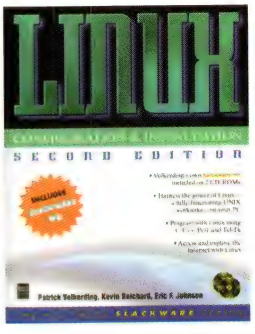
Standards For Online Communication

by JoAnn T. Hackos and Dawn M. Stevens

Guidelines for how tooplace information online within your company. Provides both a design and development process and a set of guidelines for the Internet, intranets, and help systems for designers and authors who need to create effective electionic information. Includes CD-ROM with software containing files to help you utilize the models described in the book.

Includes CD-ROM!

(BSFOC) Our Price **$40.45**



Linux Configuration & Installation, 2nd Edition

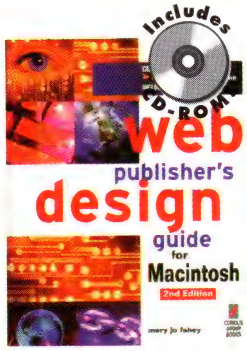
by Patrick Volkering, Kevin Reichard, and Eric F. Johnson

Linux, the leading UNIX variant, has garnered loads of attention within the UNIX community. The amazing thing about Linux is that you don't need a workstation to run it. Linux Configuration & Installation, Second Edition lets you run Linux today. Program with Linux using C, C++, Perl, and Tcl/Tk. The 2 CD-ROM pack offers one of the most popular Linux distributions, Slackware 96, and comes directly from Patrick Volkering, the creator of Slackware.

Includes CD-ROM!

(BLCI2) Our Price **$35.95**

One & 2-Day SHIPPING (call for details)



Web Publisher's Design Guide for Macintosh, 2nd Edition

by Mary Jo Fahey

This is the only book that takes you step-by-step through real projects designed by talented new media artists. Internet design experts share their design secrets and art files (look for art files on the companion CD-ROM by artist's last name). This expanded new edition includes Photoshop, Illustrator and DeBabelizer tricks from the first edition plus countless new ways to liven up your Web pages with 3D graphics, sound, movies, and much more.

(BWPDG2) Our Price **$35.99**

BASIC for the Newton

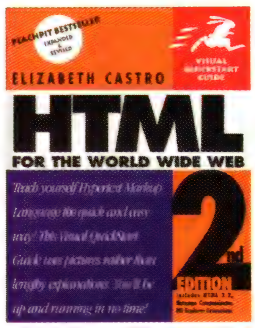
by John Schettino and Liz O'Hara

Includes Disk!

Program on Macintosh, Windows-based PC, or on the Newton itself. Straight-forward "programming by example" approach – you'll be writing Newton programs right away. Includes 3.5" disk containing Demonstration NS BASIC and over fifty example programs (Newton not included).

(BNEWT) Our Price **$32.35**

SEE RELATED CATEGORY: Dev. Environments

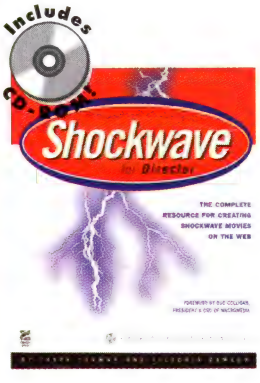


HTML For The World Wide Web, 2nd Edition

by Elizabeth Castro

Teach yourself Hypertext Markup Language the quick and easy way! This Visual QuickStart Guide uses pictures rather than lengthy explanations. You'll be up and running in no time. If you need to learn HTML fast – this is book is for you.

(BHTMLW2) Our Price **$16.15**

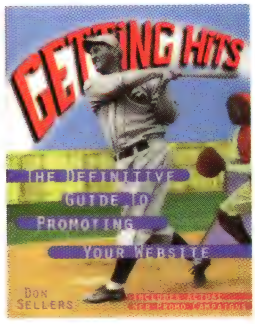


Macromedia Shockwave for Director

by Jason Yeaman and Victoria Dawson

The complete resource for creating Shockwave movies on the Web. This hands-on reference makes it easy to create Shockwave movies and put them on the Web. Expert tips from the creators of Macromedia's first Shockwave movies, together with detailed examples and instruction, provide everything you need to get started. Includes CD-ROM.

(BMSFD) Our Price **$27**

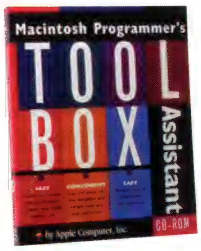


Getting Hits–The Definitive Guide To Promoting Your Website

by Don Sellers

Getting Hits explains in easy-to-understand language the underlying concepts behind the art of Web site promotion. Just a few of the topics you'll learn include: using search engines with URL's; finding related Internet groups or lists; understanding the nuances of click throughs and ad rates; and distributing press releases to key Internet contacts. With this book, you'll go beyond the conceptual and actually follow real-world tested promotional campaign strategies. Bring the world to your Web site!

(BGHITS) Our Price **$17.95**



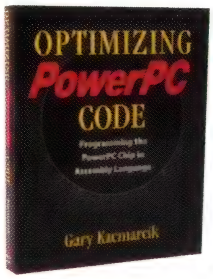
Programmer's Toolbox Assistant CD-ROM

Instant electronic access to Inside Macintosh essentials.

by Addison-Wesley Publishing

Get quick access to reference pages for over 4,000 Toolbox calls in your system software from their development environment. Essential information for Macintosh software developers. Hypertext links allow programmers to view related topics easily. The ultimate electronic reference tool for Macintosh programmers.

(STBASST) Our Price **$89.95**

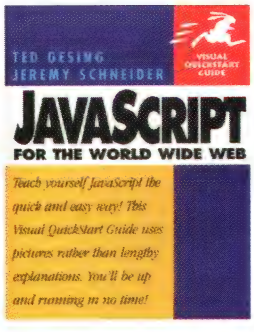


Optimizing PowerPC Code: Programming the PowerPC in Assembly Language

by Gary Kacmarcik

Take full advantage of the potential of the PowerPC by mastering the Assembly Language techniques. Learn to produce faster more robust software!

(BOPTPPC) Our Price **$35.96**



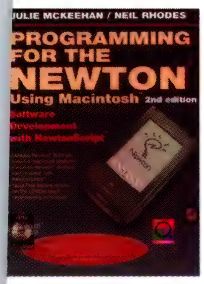
JavaScript For The World Wide Web

by Ted Gesing and Jeremy Schneider

This book takes an easy, visual approach to teaching JavaScript, where pictures guide you through the software and show you what to do. Works like a reference book, you look up what you need and then get straight to work. No long winding passages, concise, straightforward commentary explains what you need to know.

(BJWWW) Our Price **$16.15**

Check out our Web site!
• Full product descriptions • Hundreds of more products
http://www.devdepot.com



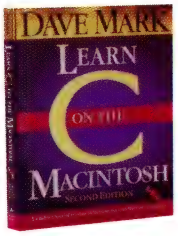
Programming for the Newton Using Macintosh:Software Development with NewtonScript- Second Edition

Includes CD-ROM!

by Julie McKeehan & Neil Rhodes

Praise for the Second Edition! "Rewritten from cover to cover, this new edition teaches you the essentials of programming for Newton 2.0. You must read this book. Then read it again. And again . . ."--SCRIBBLES (OxNUG, Australian Newton Users Group) Includes one CD-ROM for Macintosh 68030 or higher. 466 pp.

(BPFNUM) Our Price **$31.45**



Learn C on The Macintosh Second Edition

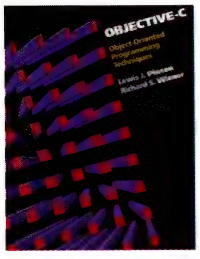
Includes CD-ROM!

by Dave Mark

New revised edition! Easy-to-understand – everything you need to start programming. Updated and enhanced exercises that lead you step by step. You'll learn function, variables, point datatypes, data structures, file input and output and more! Includes CD-ROM with Metrowerks CodeWarrior™ Lite.

(BLEARNC2) Our Price **$33.25**

SEE RELATED CATEGORY: Dev. Environments



OBJECTIVE-C Object-Oriented Programming Techniques

by Lewis J. Pinson and Richard S. Wiener

Presents the basic concepts of object-oriented design and programming, and provides a precise description of the Objective-C language. Several small-to-medium sized applications using Objective-C illustrate the general principles of object-oriented programming. Covers the two main versions of the Objective-C language. Demonstrates the versatility and power of the NeXT machine as a platform to support object-oriented programming. Shows how to design, implement, and use hierarchies of classes. Explains the purpose of pre-defined classes and shows how they can be used in designing programs.

(BOBJCOOPT) Our Price **$34.15**

Inside CodeWarrior Professional

by Metrowerks

Includes CodeWarrior IDE User's Guide. This is the printed version of the documentation provided on the CD. Covers CodeWarrior Professional Release, the debugger and associated tools.

(BINSCWP) Our Price **$34.95**

SEE RELATED CATEGORY: Dev. Environment

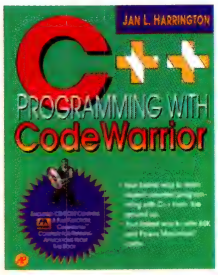
Metrowerks CodeWarrior Programming

Includes CD-ROM!

by Dan Parks Sydow

Includes CodeWarrior Lite, and Full Coverage of PowerPlant™. The best information on Metrowerks CodeWarrior, giving full coverage to the Gold Edition. CD includes Code Warrior Lite.

(BCWPROG) Our Price **$35.95**



C++ Programming with CodeWarrior

Includes CD-ROM!

by Jan L. Harrington

Beginning OOP for the Macintosh and Power Macintosh and Mac OS compatibles. Learn object-oriented programming techniques using C++ as the example language and Metrowerks and CodeWarrior as the example compiler. Enclosed CD contains example code from the book and a full-function Metrowerks CodeWarrior.

(BCPPCW) Our Price **$32.35**

CodeWarrior Software DevelopmentUsing PowerPlant

Includes CD-ROM!

by Jan L. Harrington

C++ programmers will learn to develop object-oriented software applications for the Mac and Power Mac using the PowerPlant environment and the classes that support it. Covers CodeWarrior 8. Included CD-ROM contains source code for all the programming examples in the book and Metrowerks CodeWarrior Lite.

(BCWSWDEV) Our Price **$31.45**

Inside PowerPlant

by Metrowerks

Create PowerPlant applications using the CodeWarrior IDE and PowerPlant Constructor. Full descriptions of major PowerPlant classes and resources. Included are the PowerPlant Constructor Manual, including View, TextTraits and Custom Types editing, and PowerPlant Library Reference, covering all classes and functions in PowerPlant.

(BINSPP) Our Price **$34.95**

SEE RELATED CATEGORY: Dev. Environment

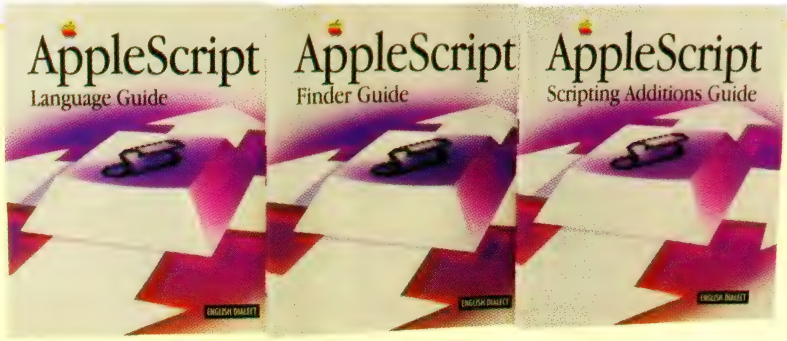
AppleScript Finder Guide, English Dialect

by Apple Computer, Inc.

Provides definitions for Finder object classes and commands. Write, record, or run scripts that trigger the same desktop actions that you trigger using the keyboard and mouse.

(BAFG) Our Price **$17.95**

SEE RELATED CATEGORY: Scripting



AppleScript Language Guide

by Apple Computer, Inc.

A complete reference for anyone using AppleScript to modify existing scripts or to write new ones. Contains useful information for programmers who are working on scriptable applications or complex scripts. Features detailed definitions of AppleScript terminology and syntax in the following categories: Value classes, commands, objects and references to objects, expressions, control statements, handlers, and script objects. Includes many sample scripts, discusses advanced topics such as writing command handlers for script applications, the scope of script variables and properties declared at different levels in a script, and inheritance and delegation among script objects.

(BALG) Our Price **$26.95**

SEE RELATED CATEGORY: Scripting

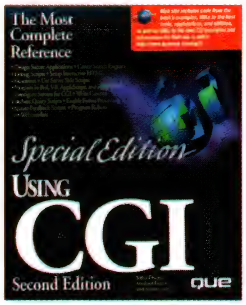
AppleScript Applications: Building Applications with FaceSpan and AppleScript

Includes CD-ROM!

by John Schettino Affiliation & Liz O'Hara

Build complete AppleScript applications using FaceSpan, a user interface development tool that makes AppleScript applications truly "Mac-Like". Uses a step-by-step approach to demonstrate techniques for building applications through illustrations and samples. Provides Graphical User Interface (GUI) design tips and practical approaches for implementation. Contains one CD-Rom with AppleScript 1.1, a demonstrations version of FaceSpan 2.1, source code for all example applications numerous AppleScript shareware and demonstrations programs. Contains a section on debugging AppleScript applications using FaceSpan.

(BAPSCAP) Our Price **$31.45**



Special Edition Using CGI, 2nd Edition

Includes CD-ROM!

by Jeffry Dwight, Michael Erwin and Robert Niles

This complete reference provides professional Web developers and advanced personal users with the latest information on using CGI (Common Gateway Interface) to interact with databases.

- Explains client and server uses of CGI
- Provides extensive coverage of live audio and video feeds, user chat and interaction, and CGI security
- Features separate chapters devoted to language-specific tips, tricks, and traps
- CD ROM is loaded with the HTML and CGI sample code from the book
- Includes applications for guest books, mail and new gateways, browser identification, access restriction, and shopping carts

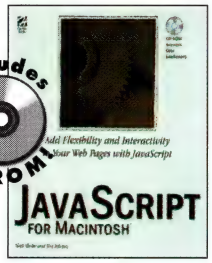
(BSEUCGI) Our Price **$44.99**

Java in a Nutshell, 2nd Edition

by David Flanagan

A detailed overview of all of the new features in Java 1.1, both on a package-by-package basis and in terms of overall functionality. A comprehensive tutorial on "inner classes" that explains how to use all of the new types of inner classes: static member classes, member classes, local classes, and anonymous classes. Practical, real-world example programs that demonstrate the new features in Java 1.1, including object serialization, the new AWT event handling model, internationalization, and a sample Java Bean.

(BJNUT2) Our Price **$17.95**



JavaScript for the Macintosh

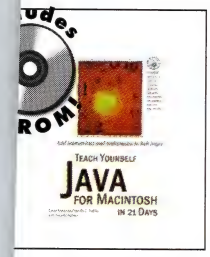
by Matt Shobe and Tim Ritchey

Allows non-programmers to take advantage of the power of Netscape Navigator. Expand the capabilities of your Web page, without having to understand C or C++. CD-ROM contains "Wizlets" that allows you to easily create your own JavaScripts. Takes you step-by-step through programming cross-platform JavaScripts. Details how to create JavaScripts for JavaScript-aware Web browsers.

(BJAVASCRPTJ) Our Price **$40.50**

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

Check out our Web site!
• Full product descriptions • Hundreds of more products
http://www.devdepot.com



Teach Yourself Java for Macintosh in 21 Days

by Laura Lemay and Charles L. Perkins with Timothy Webster

Add interactivity and multimedia to Web pages! A step-by-step guide to make your Website come alive. Learn the basics of programming Java applets and the concepts behind the Java language. Includes CD-ROM with a limited version of Roaster, the first commercial, integrated applet development environment for Java for the Macintosh!

(BJAVAMAC) Our Price **$36**

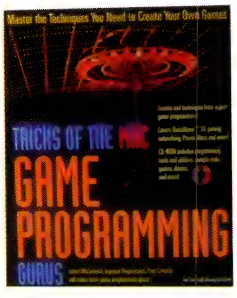
3D Graphics Programming Using QuickDraw 3D

Includes CD-ROM!

by Apple Computer, Inc.

Incorporate spectacular 3D graphics into your applications. Explore QuickDraw 3D, a revolutionary graphics extension to the Mac OS for Power Macintoshes. CD contains the complete QuickDraw 3D system itself and a complete database of the QuickDraw 3D API, allowing you instant access to the hundreds of graphics calls via a fast viewing engine. Book/CD-ROM, 640 pages.

(B3DGRAP) Our Price **$35.96**



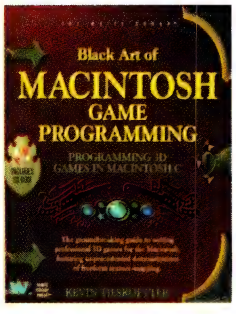
Tricks of The Mac Game Programming Gurus

Includes CD-ROM!

by McCornack, Ragnemalm, Celestin, et al.

For beginning to expert game programmers. Complete overview of all the necessary components of game programming on the Macintosh. Packed with valuable tools, utilities, sample code, CodeWarrior™ Lite and game demos. QuickDraw 3D and Power Mac optimization and inside info on how Glypha III was created. Hundreds of tried-and-true tricks, tips, and insider secrets from well-known Mac game programming experts.

(BTRICKS) Our Price **$45**



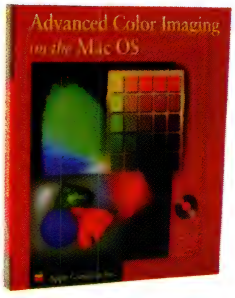
Black Art of Macintosh Game Programming

by Kevin Tieskoetter

Includes CD-ROM!

Develop your own 3D games in C on the Mac. Includes CD with project files for both Symantec C and Code Warrior. Create freeform texture-mapped games and polygon graphics. Control dynamic source code - all compatible as native to the Power Mac. Write directly to the screen, bypassing QuickDraw.

(BBLACK) Our Price **$35.99**



Advanced Color Imaging on the Mac OS

Includes CD-ROM!

by Apple Computer, Inc.

Enhance your software's color capabilities with step-by-step instructions. Augment the color support supplied with QuickDraw, and QuickDraw GX. Use the Pallette Manager to get the best colors on limited displays. Match colors between screens and input/output devices (scanners & printers). CD includes a complete reference information in both QuickView and Acrobat formats. Plus, a sample application demonstrating ColorSync programming techniques.

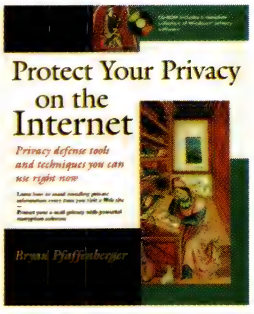
(BADVCI) Our Price **$33.25**

The Internet Marketing Plan

by Kim M. Bayne

This book gives you a comprehensive framework for producing and executing a customized Internet marketing plan. Marketing communications veteran Kim Bayne supplies you with a clear set of step-by-step procedures for establishing, implementing, evaluating, and managing your company's online presence.

(BTIMP) Our Price **$35.99**



Protect Your Privacy on the Internet

by Bryan Pfaffenberger

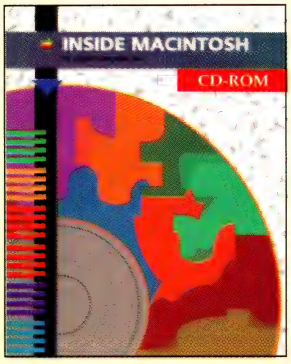
This book/CD-ROM package gives you proven privacy defense strategies and techniques to help you make the Net a safer place to work and play. You'll get the names of Internet privacy organizations that are working to protect your privacy rights and find out what you can do to help.

(BPYP) Our Price **$26.99**

Order Toll-free
800-MACDEV-1
(800-622-3381)

INSIDE MACINTOSH

by Apple Computer, Inc.



Inside Macintosh: CD-ROM

by Apple Computer, Inc.

More than 25 volumes in electronic form. Includes: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components. Access over 16,000 pages of information with Hypertext linking and extensive cross referencing.

(BIMCD) Our Price **$89**

WAIT... There's More!

Limited time offer to buy these Inside Macintosh products – 10 % off! For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	OUR PRICE
Inside Macintosh: Devices	BIMDEV	26.95
Inside Macintosh: Files	BIMFIL	26.95
Inside Macintosh: Interapplication Communications	BIMIAPP	33.25
Inside Macintosh: Memory	BIMMEM	22.45
Inside Macintosh: More Macintosh Toolbox	BIMMAC	31.45
Inside Macintosh: Networking	BIMNET	26.95
Inside Macintosh: Overview	BIMOVER	22.45
Inside Macintosh: PowerPC Numerics	BIMPPCNUM	26.05
Inside Macintosh: PowerPC System Software	BIMPPCSYS	22.45
Inside Macintosh: Processes	BIMPROC	20.65
Inside Macintosh: QuickDraw GX Prog. Overview	BIMGXOV	22.45
Inside Macintosh: QuickDraw GX Typography	BIMGXTYP	26.95
Inside Macintosh: QuickTime	BIMQT	26.95
Inside Macintosh: QuickTime Components	BIMQTCOM	31.45
Inside Macintosh: Sound	BIMSOUND	26.95
Inside Macintosh: X-Reference	BIMXREF	17.95

(Book sale prices are contingent upon availability)

For Macintosh Programmers & Developers

MacTech®
MAGAZINE

MacTech® Magazine

MacTech keeps Mac programmers & developers up to date with everything they neec know about software development. Topics like Rhapsody, Java, QuickTime, OPENSTEI Objective-C, C/C++, Object Oriented Technologies, product reviews and much more!

Subscriptions:

(MTYRDM) US/Domestic for 12 issues **$47**
(MTYRCM) Canadian for 12 issues **$59**
(MTYRFM) International for 12 issues **$97**
Back Issues: each plus shipping (subject to availability) **$10**

MacTech® CD-ROM Volumes 1-12

- Includes Apple's *develop* issues 1-29 (1990-1997)
- Almost 1600 articles from all 139 issues of MacTech Magazine (1984-1996) and through may of 1997
- Improved hypertext, improved indices, and a new THINK Reference Viewer – for lightning quick access!
- New hyperlinks between articles
- 100+ MB of source code — use them in your applications, with no royalties!
- Full version of THINK Reference™ — the original online guide to Inside Macintosh, Vols. I-VI
- 80MB of FrameWorks/SFA archives and the most complete set of FrameWorks archives known
- Sprocket™! MacTech's tiny framework that compiles quickly and supports System 7.5 features
- The best threads from the Macintosh programmer newsgroups plus thousands of notes, tips, snippets, and gotchas
- Popular tools that Macintosh programmers use to increase their productivity and much more!

(SMTCD12) Volumes 1-12 Our Price **$129**
(SMTCD12U) Upgrade from any previous version Our Price **$49**

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us. Our prices on books are at least 10% off list price.

Development Environments

PRODUCT	CODE	PRICE
AppleScript Applications: Building Applications w/FaceSpan	BAPSCAP	$31.45
Basic for the Newton - Programming using NS BASIC	BNEWT	32.35
C++ Programming	BCPPMACAP	31.46
C++ Programming with CodeWarrior	BCPPCW	32.35
C/C++ SDK User's Guide	BCPPUSER	29.00
CodeWarrior Inside PowerPlant	BINSPP	34.95
CodeWarrior Software Development using PowerPlant	BCWSWDEV	31.45
Cyberdog Programmers Kit	BCYBERDOG	34.95
Dan Shafer Presents the Power of Prograph CPX	BDANPRO	19.95
Inside CodeWarrior Book	BINSCW	34.95
Instant CORBA	BIC	17.99
Last Resort Programmers Edition	BLSTRSRT	74.95
Learn C on the Macintosh, 2nd Edition	BLEARNC2	33.25
Macintosh C Programming Primer Volume I	BCPRIM1	24.25
Macintosh C Programming Primer Volume II	BCPRIM2	24.25
Macintosh Pascal Programming Primer Volume I	BPASCPRI	24.25
Mastering the Think Class Library	BMASTERTCL	26.95
Mastering the Toolbox Using THINK C	BCPRIM2	24.25
Metrowerks CodeWarrior Programming	BCWPROG	35.95
Objective-C - Object-Oriented Programming	BOBJCOOPT	34.15
Presenting Magic Cap	BPRESMAGIC	15.25
Real World Apple Guide	BREALWLD	35.95
Symantec C++ Programming	BSYMCPP	39.50
Taligent's Guide to Designing Programs	BTALIGENT	17.55
The Power of Prograph CPX	BDANPRO	19.95
Visual Programming with Prograph CPX	BVISPRO	30.60
Wireless For The Newton	BWIRELESS	31.45

Hardware

Apple CD-ROM Handbook	BCDHAND	14.36
Designing Cards & Drivers for the Macintosh	BCARD	26.96
LaserWriter Reference	BLASERREF	17.96
PCI System Architecture 3rd Edition	BPCISYS	31.46
PowerPC System Architecture	BPPCARCH	31.46

Internet Related

1994 Internet White Pages	B94WHITE	26.95
Active Java	BACTJAVA	23.36
America Online for Dummies	BAOLDUM	17.95
CGI By Example	BCGIBE	31.45
Building & Maintaining an Intranet w/ the Macintosh	BBAMAI	45.00
Claris Home Page Companion	BCHPC	26.95
Computer Privacy Handbook	BPRIV	22.45
E-Mail Essentials	BEMAILE	22.45
Elements of E-Mail Style	BEMAIL	13.45
Hooked on Java	BHJAVA	26.95
Instant Internet Guide	BINSTANT	13.45
Internet Book	BTHENET	22.50
Internet for Dummies 2nd Edition	BNETDUM2	17.99
Internet for Dummies Quick Reference	BDUMQCK	8.05
Internet for Macs for Dummies	BNETDUM	17.95
Internet for Macs for Dummies Bestseller Edition	BIFMFDBE	35.99
Internet Power Tools	BPWRTOOL	36.00
Internet Publishing with Adobe Acrobat	BIPWAA	36.00
Internet, The, Deluxe Edition	BNETDELUX	31.50
Intranet Web Dev.: Enterprise Alternatives to Client/Server	BINTWD	44.99
Java Essentials for C/C++ Programmers	BJAVAESSEN	17.95
Java in a Nutshell	BJAVANUT	13.45
Java Language API SuperBible	BJLAS	53.99
Java Programming with CORBA	BJPWC	26.99
JavaScript for Macintosh	BJAVASCRPT	40.50
Learn HTML on the Macintosh	BLHTML	26.95
Learn Java on the Macintosh	BLJAVA	31.45
Mastering Netscape 2.0 for Macintosh, Second Edition	BMASNET2	36.00
More Internet for Dummies Starter Kit	BDUMNET	17.95
Mosaic for Dummies	BMOSDUM	17.99
Net Chat	BNETCHAT	17.00
NetObjects Fusion Handbook	BNETOFH	45.00
Netscape Navigator 3.0	BNETN3	26.96
Netscape Navigator Starter Kit for Macintosh	BNETNSK	31.49
Perl Quick Reference	BPERLREF	17.99
Planning and Managing Websites	BPLANWEB	35.96
Protect Your Privacy on the Internet	BPYP	26.99
Providing Internet Services via the MacOS	BPROVNET	31.46
Publish it on the Web	BWEBPUB	31.46
TCP/IP Vol 1-Vol 2 Bundle	BTCP12BNDL	99.00
Teach Yourself Java in 21 days	BJAVAMAC	36.00
The Internet Marketing Plan	BTIMP	35.99
Underground Guide to Telecommuting	BUNDER	22.45
Using Lotus Notes as an Intranet	BULN	40.45
Web Head Mac Guide	BWEBHEAD	22.45
Web Page Scripting Techniques	BWEBPST	45.00
Web Publishing with Adobe Acrobat and PDF	BWPWAA	35.95
Web Weaving	BWWEAV	22.45
Webmaster Macintosh	BWEBMAS	26.95

Scripting and Solutions

AppleScript Applications: Building Apps with FaceSpan	BAPSCAP	31.45
AppleScript Scripting Additions Guide	BSCRADD	17.05
Applied Macintosh Scripting	BAPPLIED	31.45
Complete HyperCard 2.2 Handbook	BHYPCRD2	31.50
Complete HyperTalk 2.2:	BHYPCRD2	31.50
Danny Goodman's Apple Guide Starter Kit	BDGAGSK	31.46
HyperCard Stack Design	BHYPSTA	19.95
JavaScript for Macintosh	BJAVASCRPT	40.50
Perl Quick Reference	BPERLREF	17.99
Real World Apple Guide	BREALWLD	35.95

Technical Reference

Active Java	BACTJAVA	23.36
Apple CD-ROM Handbook	BCDHAND	14.36
Art of Human Interface Design	BAHID	29.65
Black Art of Mac Game Programming	BBLACK	35.99
C++ for Dummies	BCPPDUM	22.46
C for Dummies Vol. 1	BCDUM	17.95
Developing Object Oriented Software for the Macintosh	BDEVOB	26.06
Extending the Mac Toolbox	BETMT	22.46
Foundations of Macintosh Programming	BFOUND	35.96
Fragment of Your Imagination	BFRAG	35.96
Guide to Macintosh Software Localization	BLOCALIZ	24.26
Guide to Macintosh System 7.5	BSYS7.5	22.
Inside AppleTalk	BAPTALK	31.
Inside the Macintosh Communications Toolbox	BCOMM	22.
LaserWriter Reference	BLASERREF	17.
Learn C++ on the Macintosh	BLRNCPP	35.
Learn C on the Macintosh, 1st Edition	BLEARNC1	31.
Learn C on the Macintosh, 2nd Edition	BLEARNC2	33.
Mac Programming for Dummies	BMACDUM	17.
Macintosh C Programmer Primer Volume 1	BCPRIM1	24.
Macintosh C Programmer Primer Volume 2	BCPRIM2	24.
Macintosh OLE2 Prog. Reference Working with Objects	BOLE2	40.
Macintosh Pascal Programming Primer Volume I	BPASCPRI	24.
Macintosh Programming Secrets 2nd edition	BPSECRET	28.
Macintosh Programming Techniques	BPTECH	31.
Microsoft Visual J++ 1.1 Sourcebook	BMVJS	35.
More Mac Programming Techniques	BMORETECH	34.
Network Frontiers Bundle	BNETFB	59.
Newton Programming Guide	BNEWTPGUID	40.
Object Oriented Programming Design	BOOPRODES	20.
Optimizing PowerPC Code	BOPTPPC	35.
Perl Quick Reference	BPERLREF	17.
PostScript Language Reference	BPSLANREF	29.
Powerbook: Digital Nomad's Guide	BPBTDNG	22.
PowerPC Programmer's Toolkit	SPPCPT	40.
Programming Introduction to the Macintosh Family	BFAMILY	22.
Programming for System 7	BSYS7	24.
Programming Primer Macintosh Volume 1	BPRIMMAC	34.
Programming with AppleTalk	BPROAT	22.
QuickTime - Official Guide for Macintosh Users	BQTGUIDE	45.
Real World Apple Guide	BREALWLD	35.
ResEdit All Night Diner	BRESDINE	22.
ResEdit Complete, 2nd Edition	BRESED2	31.
ResEdit Reference	BRESEDREF	26.
Software by Design: Creating User Friendly Software	BDESIGN	26.
Symantec C++ for the Macintosh: The Basics	BSCFTMTB	31.
Teach Yourself Macintosh C++ in 21 Days	BCPP21D	26.
Technical Introduction to the Macintosh Family	BTITTMF	24.
Tog on Software Design	BTOG	26.
Wireless for the Newton Development for Mobil Comm	BWIRELESS	31.
Writing Localizable Software	BLOCAL	24.

Miscellaneous

3D Graphics-Tips, Tricks, & Techniques	B3DGTTT	31.46
A Fragment of Your Imagination	BFRAG	35.96
Adobe Premiere for the Macintosh	BPREM	44.95
America Online for Dummies	BAOLDUM	17.95
AppleGuide Complete	BAPLGD	35.96
Art of Human Interface Design	BAHID	35.96
CD-ROM Guide to Multimedia Authoring	BCDMULTI	40.45
CompuServe for Dummies	BCSDUM	17.95
Creating Interactive CD-ROM	BINTERCDR	35.95
Cyberpunk Handbook	BCYBPUNK	8.95
Danny Goodman's Apple Guide Starter Kit	BDGAGSK	31.46
Danny Goodman's Macintosh Handbook	BGOODHB	26.95
FrameWorks Source Code Disk	MTFWSC	9.95
FrameWorks Magazine Back Issue	MTFWBACK	8.00
Global Interface Design	BGLOBAL	32.35
Graphic Gems 2	BGEMS2	44.95
Graphic Gems 4	BGEMS4	44.95
Graphic Gems V	BGEMS5	44.95
Infini-D Revealed	BINFDREV	40.50
Inside Director 5 with Lingo for Macintosh	BID5WLM	44.99
Late Night with MacHack	BLATE	26.95
Mac Bathroom Reader	BBATH	11.70
Macromedia Director Lingo Workshop, 2nd Edition	BMDLW2	40.50
Mac Screamer: The Ultimate Macintosh Supercharging Kit	BSCREAM	31.50
Macintosh Crash Course	BCRASH	26.95
MacTech Back Issues	MTBACKISS	10.00
Macworld Ultimate Macintosh Programming Book	BULTMAC	35.95
Managing AppleShare & Workgroup Servers	BMAWS	26.95
MADACON '93 CD-ROM	SMADA93	9.95
Multimedia Authoring: Building and Developing Documents	BMMAUTH	31.45
Multimedia Starter Kit for Macintosh	BMMSTART	27.00
Network Frontiers Bundle	BNETFB	59.95
Profit from Experience	BPROFIT	22.45
ResEdit Complete, Second Edition	BRESED2	31.45
Sad Macs, Bombs and Disasters	BSADMAC	22.45
Stupid Mac Tricks	BSTUPIDMAC	17.95
The Elements of E-Mail Style	BEMAIL	13.45
The Software Developer's & Marketer's Legal Companion	BSDAMLC	33.26
Tog on Software Design	BTOG	26.95
Tricks of the Mac Game Gurus	BTRICKS	45.00
Zen and the Art of Resource Editing	BZAAORE	27.00

All entries in this index are alphabetized. For your convenience the product names are **bold**, book names are *italicized* and company names are in plain text.

INDEX

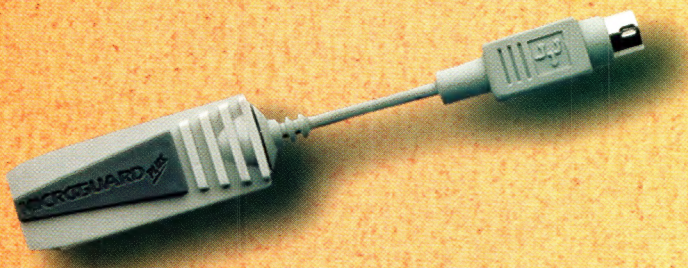
INDEX

REWARD
$15,000,000,000

Have you seen this man ?
Hacker Harry and the pirate gang are loose and robbing the software stagecoach. Whether it's software, bank account information, money transactions, or customer databases, if you want to stop them and collect your part of the reward... the place to find your New Generation Protection solution is at:

www.micromacro.com

The expected losses due to piracy in the software industry alone are over $15 billion for 1997. Some of this belongs to YOU. By using MicroGuard's New Generation Copy Protection keys, you will enforce your rights and guarantee your profits.



International
Micro Macro Technologies, Ltd.
3 Hashikma St.
P.O.Box 11516, Azur 58001
Israel

Tel: (972-3) 558-2345
Fax: (972-3) 558-2344
E-mail: info@micromacro.com

USA
MicroGuard
631 South Pontiac St.
Denver, CO 80224
USA

Tel: (303) 320-1628
Fax: (303) 320-1599
E-mail: usa@micromacro.com

SEE YOU AT:

COMDEX Fall '97
November 17-21, 1997
LAS VEGAS

Booth #S8049

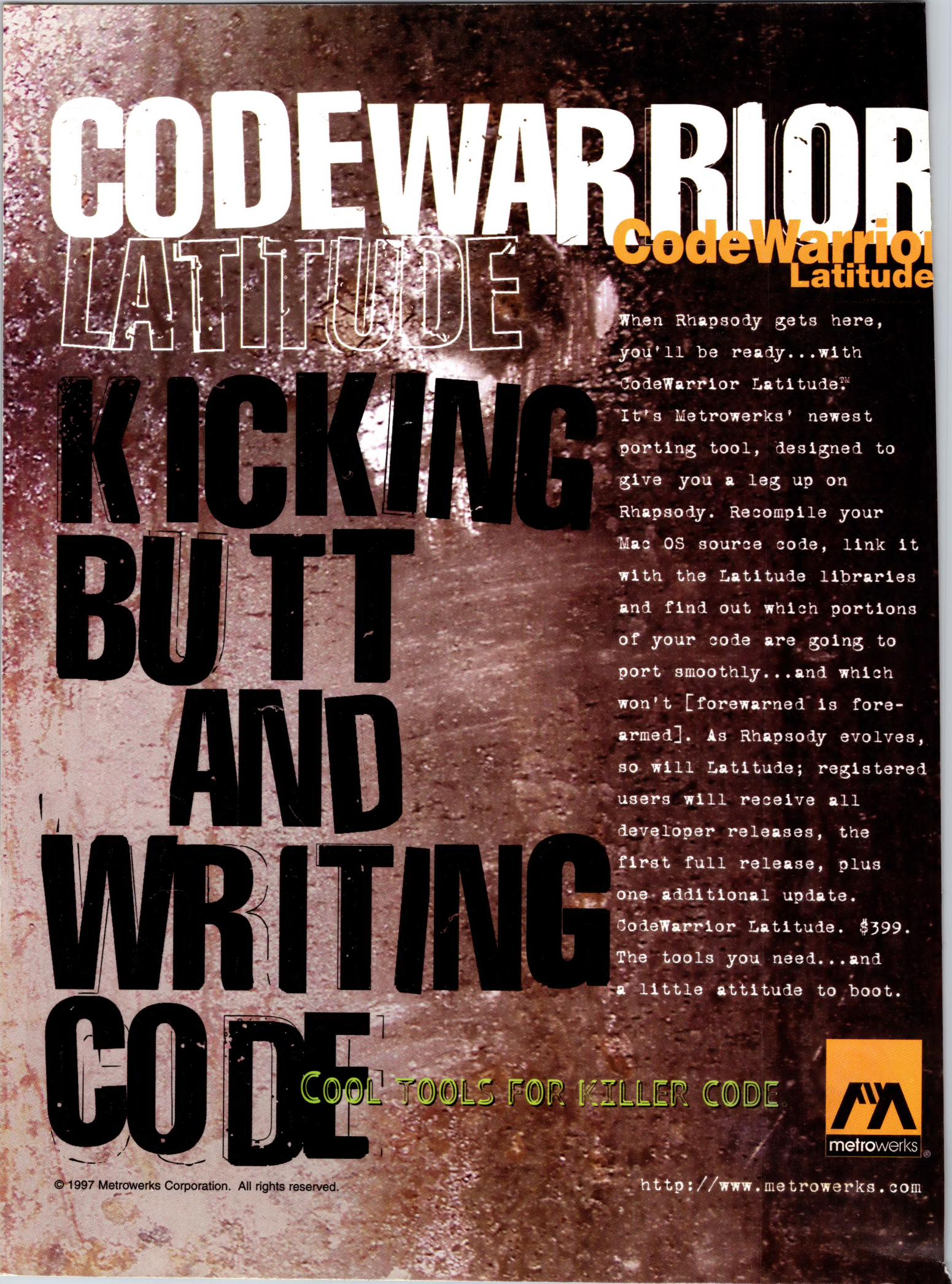
MICROSOFT® WINDOWS™ COMPATIBLE

Mac™OS

Charter member Pirate Busters

MICROGUARD™

©MicroGuard and MicroGuard PC are registered trademarks of Micro Macro Technologies, Ltd. All other product names are trademarks of their respective owners. Mac and the Mac O/S logo are trademarks of Apple Computer, Inc., Windows and Flying Windows logo are trademarks of Microsoft Corporation.


CODEWARRIOR
LATITUDE
KICKING
BUTT
AND
WRITING
CODE
CodeWarrior
Latitude
When Rhapsody gets here, you'll be ready...with CodeWarrior Latitude.™ It's Metrowerks' newest porting tool, designed to give you a leg up on Rhapsody. Recompile your Mac OS source code, link it with the Latitude libraries and find out which portions of your code are going to port smoothly...and which won't [forewarned is fore-armed]. As Rhapsody evolves, so will Latitude; registered users will receive all developer releases, the first full release, plus one additional update. CodeWarrior Latitude. $399. The tools you need...and a little attitude to boot.
COOL TOOLS FOR KILLER CODE.
metrowerks
© 1997 Metrowerks Corporation. All rights reserved.
http://www.metrowerks.com